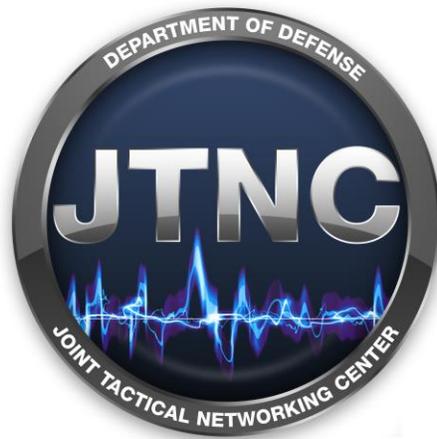


**Joint Tactical Radio System Standard  
Serial Port Device  
Application Program Interface**



**Version: 2.1.4  
26 June 2013**

Statement A- Approved for public release; distribution is unlimited (17 July 2013).

## Revision History

Version	Description	Last Modified Date
1.0	Initial release <b>ICWG Approved</b>	23-Jan-2006
1.1	Update outline format <b>ICWG Approved</b>	02-Feb-2006
1.2	Updates from Increment 4 ICWG; Added appendices for payload size specification and HDLC properties extension. <b>ICWG Approved</b>	09-May-2006
1.3	Updates from submitted SAR to add tx and rx HDLC headers.	13-Oct-2006
1.4	Removed BYTE_ALIGNED from SyncClockControlType	15-Oct-2006
1.5	Revised Packet Consumer and Producer Interfaces	26-Oct-2006
1.6	Updates based on Increment 6 comments	06-Dec-2006
2.0	Updated Packet Interfaces; Updated to camelback naming conventions. <b>ICWG Approved</b>	13-Dec-2006
2.0.1	Preparation for public release	29-Mar-2007
2.0.2	Errata: In D.4.1 “SerialSyncHdlcExt.idl” -Corrected value of CONFIG_SYNC_HDLC -Changed “tx/rxAltFrameFillLength” to “tx/rxAltFrameFill” -Changed “syncRawConfiguration” to “syncHdlcConfiguration”  -Moved all payload size specifications from A.2.1, A.2.2, and A.2.3 to Appendix A.C	02-Apr-2008
2.1 <Draft>	SAR: -Added missing fields (i.e. “enableAutoBaud”, “enableRtsCts”, and “enableCdpMode”) to the <i>SyncHdlcConfigType</i> structure  Change Proposals: -Replaced valid range specific baud rates with baud rate specifications (i.e. Appendix B.C, Appendix C.C, and Appendix D.C) in <i>AsyncConfigType</i> ,	20-Aug-2007

Version	Description	Last Modified Date
	<p><i>SyncRawConfigType</i>, and <i>SyncHdlcConfigType</i> structures.</p> <ul style="list-style-type: none"> <li>-Removed “stop” statement from description in <i>setAsyncConfig</i>, <i>setSyncRawConfig</i>, and <i>setSyncHdlcConfig</i> operations.</li> <li>-Added Section E, Clock Adjustment Extension</li> </ul>	
2.1	Updates based on ICWG review	01-Oct-2007
2.1.1 <Draft>	<p>Change Proposals:</p> <ul style="list-style-type: none"> <li>- Changed “boolean enableAutoBaud” to “octet autoBaudLimit” in <i>SyncHdlcConfig</i> and <i>SyncRawConfig</i> structures and added automatic baud rate limit specifications (i.e. Appendix C.D and Appendix D.D)</li> <li>- Replaced Clock Adjustment Extension with Dynamic Baud Extension.</li> </ul>	21-Feb-2008
2.1.1	<ul style="list-style-type: none"> <li>- Updates based on ICWG review</li> <li>- Removed Service State applicability</li> <li>- Incorporated minor “API” textual changes</li> <li>- Preparation for public release</li> </ul> <p><b>ICWG Approved</b></p>	30-Apr-2008
2.1.2	<p>Preparation for public release</p> <p><b>ICWG Approved</b></p>	29-May-2009
2.1.3 <Draft>	<p>A.2.2 Provides Properties API: Added descriptions for SerialPortEcho and SerialASCII</p> <p>A.2.5.1.1Device Provides Data Sequence Using Flow Resume and RTS/CTS Flow Control: Added descriptions to specify RTS/CTS signals may be tied to RTS/CTS HW signals.</p> <p>A.2.5.1.2Device Consumes Data Sequence Using Flow Resume and RTS/CTS Flow Control: Added descriptions to specify RTS/CTS signals may be tied to RTS/CTS HW signals.</p> <p>A.3.4.4<i>setLoopback</i> Operation: Added descriptions “internal Serial Port HW loopback pin.”</p> <p>A.3.4.6<i>setProtocolType</i> Operation: Added “physical” to description.</p> <p>A.5.1.1SerialPort::SynchronizationConfig: Added descriptions to “SynchronizationConfig”</p> <p>A.5.2.1SerialPort::ProtocolTypes: Added descriptions “to configure the voltage level on the Serial Port Device HW.”</p> <p>B.5.4.1Async::AsyncConfigType: Added description to swFlowControl and added notes for clarification.</p>	23-Jul-2010

Version	Description	Last Modified Date
	<p>C.5.4.1 SyncRaw::SyncRawConfigType: Added descriptions to autoBaudLimit, clockCtrl, baudRate, and enableRtsCts. Added notes for clarification.</p> <p>D.5.2.3 SyncHdlc::SyncClockControlType Enumeration: Added descriptions to RTS_CTS</p> <p>D.5.4.2 SyncHdlc::SyncHdlcConfigType Structure: Added descriptions to clockCtrl, autoBaudLimit, and enableRtsCts. Added notes for clarification.</p> <p>Figure 8 – Device Consumes Data Sequence Diagram Using Flow Resume and RTS/CTS Flow Control: Update figure for clarification</p> <p>-Added following sections for usage clarification</p> <p>A.3.1.1 Packet::PayloadStatus::getDesiredPayloadSize Operation</p> <p>A.3.1.2 Packet::PayloadStatus::getMinOverrideTimeout Operation</p> <p>A.3.1.3 DeviceIo::DeviceIoControl::enableRtsCts Operation</p> <p>A.3.1.4 DeviceIo::DeviceIoControl::setRts Operation</p> <p>A.3.2.1 Packet::PayloadStatus::setDesiredPayloadSize Operation</p> <p>A.3.2.2 Packet::PayloadStatus::setMinOverrideTimeout Operation</p> <p>A.3.2.3 DeviceIo::DeviceIoSignals::setCts Operation</p>	
2.1.3 <Final Draft>	<p>Section A.1 add description</p> <p>Section A.2.5.1.1 Update description to match changes to Figures 7</p> <p>Update Figure 7 (Changes direction of enableRtsCts)</p> <p>Section A.2.5.1.2 Update description to match changes to Figure 8</p> <p>Update Figure 8 (Remove enableRtsCts, setRts, setCts)</p> <p>Add assumption note in section: A.2.1 and A.2.3</p> <p>Gray out enableRtsCts in A.2.3</p> <p>Update port names in Section A.1.2.1</p> <p>Update port names in Section A.2.1</p> <p>Update port names in Section A.2.3</p> <p>Update port names in Appendix A.B</p>	17-Sep-2010

---

Version	Description	Last Modified Date
	Update port names in Appendix A.C	
2.1.3	- Updates based on ICWG review - In Table 1 – Serial Port Device API Provide Service Interface change getMaxPayloadSize() and getMinPayloadSize() from gray to black and getDesiredPayloadSize() and getMinOverrideTimeout() from black to gray. <b>ICWG Approved</b>	28-Sep-2010
2.1.4	Preparation for public release <b>ICWG Approved</b>	26-Jun-2013

## Table of Contents

<b>A. SERIAL PORT DEVICE API .....</b>	<b>14</b>
<b>B. ASYNCHRONOUS EXTENSION .....</b>	<b>48</b>
<b>C. SYNCHRONOUS RAW EXTENSION .....</b>	<b>60</b>
<b>D. SYNCHRONOUS HDLC EXTENSION .....</b>	<b>71</b>
<b>E. DYNAMIC BAUD EXTENSION .....</b>	<b>85</b>

---

## Table of Contents

<b>A. SERIAL PORT DEVICE API .....</b>	<b>14</b>
<b>A.1 Introduction.....</b>	<b>14</b>
A.1.1 Overview .....	14
A.1.2 Service Layer Description .....	14
A.1.2.1 Serial Port Device API Port Connections .....	14
A.1.3 Modes of Service.....	15
A.1.4 Service States .....	15
A.1.5 Referenced Documents.....	15
A.1.5.1 Government Documents .....	16
A.1.5.2 Specifications .....	16
A.1.5.3 Other Government Agency Documents .....	16
A.1.5.4 Commercial Standards .....	16
<b>A.2 Services .....</b>	<b>17</b>
A.2.1 Provide Services .....	17
A.2.2 Provides Properties API .....	20
A.2.3 Use Services .....	21
A.2.4 Interface Modules .....	23
A.2.4.1 SerialPort.....	23
A.2.5 Sequence Diagrams .....	26
A.2.5.1 Transfer Data.....	26
<b>A.3 Service Primitives and Attributes .....</b>	<b>30</b>
A.3.1 SerialPort::SerialPortPacketConsumer.....	30
A.3.1.1 Packet::PayloadStatus::getDesiredPayloadSize Operation.....	30
A.3.1.2 Packet::PayloadStatus::getMinOverrideTimeout Operation .....	30
A.3.1.3 DeviceIo::DeviceIoControl::enableRtsCts Operation.....	30
A.3.1.4 DeviceIo::DeviceIoControl::setRts Operation .....	30
A.3.2 SerialPort::SerialPortPacketProducer.....	30
A.3.2.1 Packet::PayloadStatus::setDesiredPayloadSize Operation .....	30
A.3.2.2 Packet::PayloadStatus::setMinOverrideTimeout Operation .....	30
A.3.2.3 DeviceIo::DeviceIoSignals::setCts Operation.....	31
A.3.3 SerialPort::SerialMessageControl .....	31
A.3.4 SerialPort::SerialPortConfiguration .....	31
A.3.4.1 setSynchronizationConfig Operation .....	31
A.3.4.2 getSynchronizationConfig Operation.....	32
A.3.4.3 getSynchronizationConfigsSupported Operation.....	33
A.3.4.4 setLoopback Operation .....	34
A.3.4.5 getLoopback Operation .....	35
A.3.4.6 setProtocolType Operation .....	36
A.3.4.7 getProtocolType Operation .....	37
<b>A.4 IDL .....</b>	<b>38</b>
A.4.1 SerialPort .....	38
<b>A.5 UML .....</b>	<b>41</b>
A.5.1 Data Types.....	42

A.5.1.1 SerialPort::SynchronizationConfig .....	42
A.5.1.2 SerialPort::SynchronizationConfigSequence .....	42
A.5.2 Enumerations .....	42
A.5.2.1 SerialPort::ProtocolTypes .....	42
A.5.3 Exceptions .....	42
A.5.4 Structures .....	43
<b>Appendix A.A – Abbreviations and Acronyms.....</b>	<b>44</b>
<b>Appendix A.B – Performance Specification.....</b>	<b>45</b>
<b>Appendix A.C – Payload Size Specification.....</b>	<b>46</b>
<b>B. ASYNCHRONOUS EXTENSION .....</b>	<b>48</b>
<b>B.1 Introduction.....</b>	<b>48</b>
B.1.1 Overview .....	48
B.1.2 Service Layer Description.....	48
B.1.2.1 Asynchronous Serial Port Device API Extension Port Connections.....	48
B.1.3 Modes of Service .....	49
B.1.4 Service States.....	49
B.1.5 Referenced Documents.....	49
<b>B.2 Services.....</b>	<b>50</b>
B.2.1 Provide Services .....	50
B.2.2 Use Services .....	50
B.2.3 Interface Modules .....	50
B.2.3.1 SerialPort.....	50
B.2.4 Sequence Diagrams .....	50
<b>B.3 Service Primitives and Attributes.....</b>	<b>51</b>
B.3.1 SerialPort::Async.....	51
B.3.1.1 <i>setAsyncConfig</i> Operation .....	51
B.3.1.2 <i>getAsyncConfig</i> Operation.....	52
<b>B.4 IDL .....</b>	<b>53</b>
B.4.1 SerialAsyncExt .....	53
<b>B.5 UML .....</b>	<b>55</b>
B.5.1 Data Types .....	56
B.5.2 Enumerations .....	56
B.5.2.1 Async::NumberDataBitsType .....	56
B.5.2.2 Async::ParitySet .....	56
B.5.2.3 Async::StopBitsType.....	56
B.5.2.4 SerialPort::SynchronizationConfig .....	57
B.5.3 Exceptions .....	57
B.5.4 Structures .....	57
B.5.4.1 Async::AsyncConfigType .....	57
<b>Appendix B.A – Abbreviations and Acronyms.....</b>	<b>59</b>
<b>Appendix B.B – Performance Specification.....</b>	<b>59</b>
<b>Appendix B.C – Baud Rate Specification.....</b>	<b>59</b>
<b>C. SYNCHRONOUS RAW EXTENSION .....</b>	<b>60</b>

<b>C.1 Introduction</b> .....	<b>60</b>
C.1.1 Overview .....	60
C.1.2 Service Layer Description .....	60
C.1.2.1 Synchronous Raw Serial Port Device API Extension Port Connections .....	60
C.1.3 Modes of Service .....	61
C.1.4 Service States.....	61
C.1.5 Referenced Documents.....	61
<b>C.2 Services</b> .....	<b>62</b>
C.2.1 Provide Services .....	62
C.2.2 Use Services .....	62
C.2.3 Interface Modules .....	62
C.2.3.1 SerialPort.....	62
C.2.4 Sequence Diagrams .....	62
<b>C.3 Service Primitives and Attributes</b> .....	<b>63</b>
C.3.1 SerialPort::SyncRaw.....	63
C.3.1.1 <i>setSyncRawConfig</i> Operation.....	63
C.3.1.2 <i>getSyncRawConfig</i> Operation .....	64
<b>C.4 IDL</b> .....	<b>65</b>
C.4.1 SerialSyncRawExt .....	65
<b>C.5 UML</b> .....	<b>67</b>
C.5.1 Data Types .....	68
C.5.2 Enumerations .....	68
C.5.2.1 SerialPort::SynchronizationConfig .....	68
C.5.2.2 SyncRaw::ClockSources .....	68
C.5.2.3 SyncRaw::SyncClockControlType.....	68
C.5.3 Exceptions .....	68
C.5.4 Structures .....	69
C.5.4.1 SyncRaw::SyncRawConfigType .....	69
<b>Appendix C.A – Abbreviations and Acronyms</b> .....	<b>70</b>
<b>Appendix C.B – Performance Specification</b> .....	<b>70</b>
<b>Appendix C.C – Baud Rate Specification</b> .....	<b>70</b>
<b>Appendix C.D – Automatic Baud Rate Limit Specification</b> .....	<b>70</b>
<b>D. SYNCHRONOUS HDLC EXTENSION</b> .....	<b>71</b>
<b>D.1 Introduction</b> .....	<b>71</b>
D.1.1 Overview .....	71
D.1.2 Service Layer Description .....	71
D.1.2.1 Synchronous HDLC Serial Port Device API Extension Port Connections.....	71
D.1.3 Modes of Service.....	72
D.1.4 Service States .....	72
D.1.5 Referenced Documents.....	72
<b>D.2 Services</b> .....	<b>73</b>
D.2.1 Provide Services .....	73
D.2.2 Use Services .....	73
D.2.3 Interface Modules.....	73

D.2.3.1 SerialPort.....	73
D.2.4 Sequence Diagrams .....	73
<b>D.3 Service Primitives and Attributes .....</b>	<b>74</b>
D.3.1 SerialPort::SyncHdlc .....	74
D.3.1.1 <i>setSyncHdlcConfig</i> Operation .....	74
D.3.1.2 <i>getSyncHdlcConfig</i> Operation.....	75
<b>D.4 IDL .....</b>	<b>76</b>
D.4.1 SerialSyncHdlcExt .....	76
<b>D.5 UML .....</b>	<b>79</b>
D.5.1 Data Types.....	80
D.5.2 Enumerations.....	80
D.5.2.1 SerialPort::SynchronizationConfig .....	80
D.5.2.2 SyncHdlc::ClockSources.....	80
D.5.2.3 SyncHdlc::SyncClockControlType Enumeration .....	80
D.5.3 Exceptions .....	80
D.5.4 Structures.....	81
D.5.4.1 SyncHdlc::HdlcHeaderType Structure.....	81
D.5.4.2 SyncHdlc::SyncHdlcConfigType Structure .....	81
<b>Appendix D.A – Abbreviations and Acronyms.....</b>	<b>83</b>
<b>Appendix D.B – Performance Specification.....</b>	<b>83</b>
<b>Appendix D.C – Baud Rate Specification.....</b>	<b>83</b>
<b>Appendix D.D – Automatic Baud Rate Limit Specification .....</b>	<b>83</b>
<b>E. DYNAMIC BAUD EXTENSION .....</b>	<b>85</b>
<b>E.1 Introduction.....</b>	<b>85</b>
E.1.1 Overview.....	85
E.1.2 Service Layer Description.....	85
E.1.2.1 Dynamic Baud Serial Port Device API Extension Port Connections.....	85
E.1.3 Modes of Service .....	86
E.1.4 Service States .....	86
E.1.5 Referenced Documents .....	86
<b>E.2 Services.....</b>	<b>87</b>
E.2.1 Provide Services .....	87
E.2.2 Use Services.....	87
E.2.3 Interface Modules .....	87
E.2.3.1 SerialPort .....	87
E.2.4 Sequence Diagrams.....	87
<b>E.3 Service Primitives and Attributes.....</b>	<b>88</b>
E.3.1 SerialPort::DynamicBaud .....	88
E.3.1.1 <i>setBaudRate</i> Operation.....	88
E.3.1.2 <i>getEffectiveBaudRate</i> Operation.....	89
<b>E.4 IDL .....</b>	<b>90</b>
E.4.1 SerialDynBaudExt .....	90
<b>E.5 UML .....</b>	<b>91</b>
E.5.1 Data Types .....	92

---

E.5.2 Enumerations .....	92
E.5.2.1 DynamicBaud::DataFlow .....	92
E.5.3 Exceptions.....	92
E.5.4 Structures .....	92
<b>Appendix E.A – Abbreviations and Acronyms.....</b>	<b>93</b>
<b>Appendix E.B – Performance Specification .....</b>	<b>93</b>

---

## Lists of Figures

FIGURE 1 – SERIAL PORT DEVICE PORT DIAGRAM.....	15
FIGURE 2 – SERIAL PORT DEVICE API CLASS DIAGRAM.....	23
FIGURE 3 – SERIALPORTCONFIGURATION INTERFACE.....	24
FIGURE 4 – SERIALPORTPACKETCONSUMER INTERFACE .....	24
FIGURE 5 – SERIALPORTPACKETPRODUCER INTERFACE .....	25
FIGURE 6 – SERIALMESSAGECONTROL INTERFACE.....	25
FIGURE 7 – DEVICE PROVIDES DATA SEQUENCE DIAGRAM USING FLOW RESUME AND RTS/CTS FLOW CONTROL.....	27
FIGURE 8 – DEVICE CONSUMES DATA SEQUENCE DIAGRAM USING FLOW RESUME AND RTS/CTS FLOW CONTROL.....	29
FIGURE 9 – SERIAL PORT DEVICE COMPONENT DIAGRAM .....	41
FIGURE 10 – SERIAL PORT DEVICE API ASYNCHRONOUS EXTENSION PORT DIAGRAM .....	49
FIGURE 11 – ASYNC INTERFACE CLASS DIAGRAM .....	50
FIGURE 12 – ASYNCHRONOUS EXTENSION COMPONENT DIAGRAM .....	55
FIGURE 13 – SERIAL PORT DEVICE API SYNCHRONOUS RAW EXTENSION PORT DIAGRAM .....	61
FIGURE 14 – SYNCRAW INTERFACE CLASS DIAGRAM.....	62
FIGURE 15 – SYNCHRONOUS RAW EXTENSION COMPONENT DIAGRAM .....	67
FIGURE 16 – SERIAL PORT DEVICE API SYNCHRONOUS HDLC EXTENSION PORT DIAGRAM .....	72
FIGURE 17 – SYNCHDLC INTERFACE CLASS DIAGRAM .....	73
FIGURE 18 – SYNCHRONOUS HDLC EXTENSION COMPONENT DIAGRAM .....	79
FIGURE 19 – SERIAL PORT DEVICE API DYNAMIC BAUD EXTENSION PORT DIAGRAM .....	86
FIGURE 20 – DYNAMIC BAUD INTERFACE CLASS DIAGRAM.....	87
FIGURE 21 – DYNAMIC BAUD EXTENSION COMPONENT DIAGRAM .....	91

## List of Tables

TABLE 1 – SERIAL PORT DEVICE API PROVIDE SERVICE INTERFACE .....	17
TABLE 2 – SERIAL PORT DEVICE PROVIDE PROPERTIES .....	20
TABLE 3 – SERIAL PORT DEVICE API USES SERVICE INTERFACE .....	21
TABLE 4 – SERIAL PORT DEVICE PERFORMANCE SPECIFICATION .....	45
TABLE 5 – PAYLOAD SIZE PROPERTIES PROVIDE SERVICE INTERFACE .....	46
TABLE 6 – PAYLOAD SIZE PROVIDES SERVICE INTERFACE .....	46
TABLE 7 – PAYLOAD SIZE USES SERVICE INTERFACE .....	47
TABLE 8 – ASYNCHRONOUS EXTENSION PROVIDE SERVICE INTERFACE .....	50
TABLE 9 – SERIAL PORT DEVICE PERFORMANCE SPECIFICATION .....	59
TABLE 10 – BAUD RATE SPECIFICATION .....	59
TABLE 11 – SYNCHRONOUS RAW EXTENSION PROVIDE SERVICE INTERFACE .....	62
TABLE 12 – SERIAL PORT DEVICE PERFORMANCE SPECIFICATION .....	70
TABLE 13 – BAUD RATE SPECIFICATION .....	70
TABLE 14 – AUTOMATIC BAUD RATE LIMIT SPECIFICATION .....	70
TABLE 15 – SYNCHRONOUS HDLC EXTENSION PROVIDE SERVICE INTERFACE .....	73
TABLE 16 – SERIAL PORT DEVICE PERFORMANCE SPECIFICATION .....	83
TABLE 17 – BAUD RATE SPECIFICATION .....	83
TABLE 18 – AUTOMATIC BAUD RATE LIMIT SPECIFICATION .....	84
TABLE 19 – DYNAMIC BAUD EXTENSION PROVIDE SERVICE INTERFACE .....	87
TABLE 20 – SERIAL PORT DEVICE PERFORMANCE SPECIFICATION .....	93

## A. SERIAL PORT DEVICE API

### A.1 INTRODUCTION

A *Serial Port Device* component realizes the Serial Port Device API and supports methods and attributes specific to the Serial Port hardware (HW) device it represents. The *Serial Port Device* API provides the ability to send flow controlled *octet* packets to/from the device or service user and supports Request To Send (RTS) / Clear To Send (CTS) handshaking. Serial Port communications are not symmetrical. RTS/CTS operations are dependent on whether the device is a Data Terminal Equipment (DTE) or Data Communications Equipment (DCE).

The *Serial Port Device* API also provides a base configuration interface. It should be noted that the base *Serial Port Device* API requires that a Serial Port Device component realizes at least one of the extensions (see B.1, C.1, D.1). The configuration of each synchronization mode (e.g. asynchronous, synchronous raw or synchronous High-Level Data Link Control (HDLC)) will be performed within each extension.

This document defines a common set of Serial Port Device component provide services and interfaces required by most Joint Tactical Radio (JTR) Sets.

#### A.1.1 Overview

This base contains as follows:

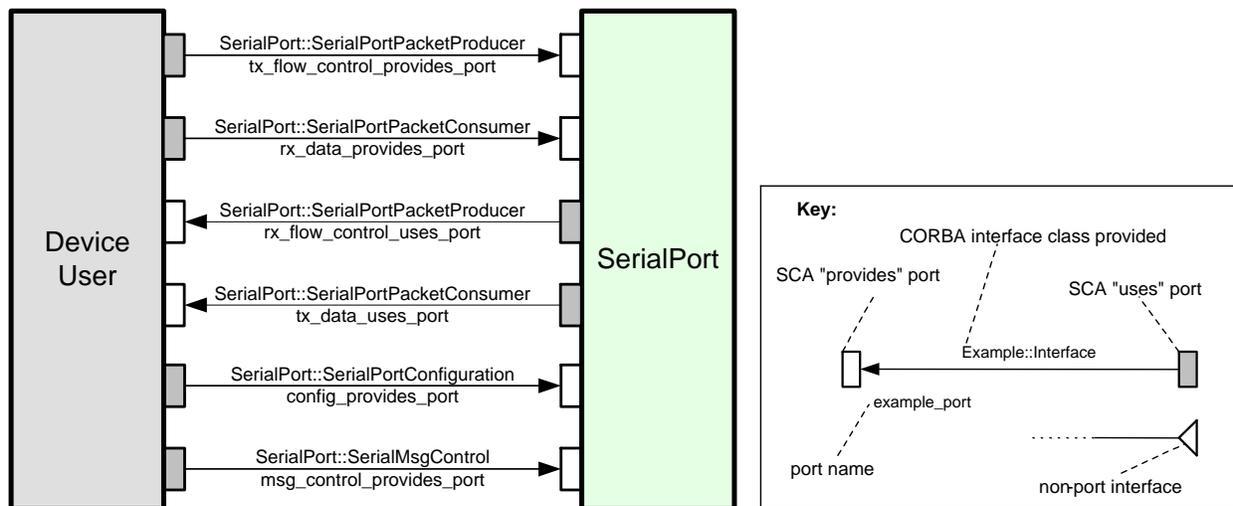
- a. Section A.1, *Introduction*, contains the introductory material regarding the overview, service layer description, modes, states and referenced documents of this document.
- b. Section A.2, *Services*, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.
- c. Section A.3, *Service Primitives and Attributes*, specifies the operations that are provided by the *Serial Port Device API*.
- d. Section A.4, *IDL*.
- e. Section A.5, *UML*.
- f. Appendix A.A, – *Abbreviations and Acronyms*.
- g. Appendix A.B, – *Performance Specification*.
- h. Appendix A.C, – *Payload Size Specification*

#### A.1.2 Service Layer Description

##### A.1.2.1 Serial Port Device API Port Connections

Figure 1 shows the port connections for a *Serial Port Device* component that realizes the *Serial Port Device API*

Note: All port names are for reference only.



**Figure 1 – Serial Port Device Port Diagram**

### ***Serial Port Device API Provides Ports Definitions***

**rx\_data\_provides\_port** is provided by the *Serial Port Device* to consume flow controlled data to/from the *Device User*. It also enables/initiates Request To Send (RTS) / Clear To Send (CTS) handshaking.

**tx\_flow\_control\_provides\_port** is provided by the *Serial Port Device* to produce packets and provide flow control signals. It also signals a CTS condition.

**config\_provides\_port** is provided by the *Serial Port Device* to set and get the serial port configuration values.

**msg\_control\_provides\_port** is provided by the *Serial Port Device* to control the device data flows.

### ***Serial Port Device API Uses Ports Definitions***

**tx\_data\_uses\_port** is used by the *Serial Port Device* to consume flow controlled data to/from the *Device User*. It also enables/initiates RTS/CTS handshaking.

**rx\_flow\_control\_uses\_port** is used by the *Serial Port Device* to produce packets and provide flow control signals. It also signals a CTS condition.

## **A.1.3 Modes of Service**

Please see section A.1 Introduction.

## **A.1.4 Service States**

Not applicable

## **A.1.5 Referenced Documents**

The following documents of the exact issue shown form a part of this specification to the extent specified herein.

### **A.1.5.1 Government Documents**

The following documents are part of this specification as specified herein.

### **A.1.5.2 Specifications**

#### **A.1.5.2.1 Federal Specifications**

None

#### **A.1.5.2.2 Military Specifications**

None

### **A.1.5.3 Other Government Agency Documents**

- [1] JTRS Standard, "JTRS CORBA Types," JPEO, Version 1.0.2.
- [2] JTRS Standard, "Packet API," JPEO, Version 2.0.2.
- [3] JTRS Standard, "DeviceIO API," JPEO, Version 1.0.2.
- [4] JTRS Standard, "Device Message Control API," JPEO, Version 1.1.3.

### **A.1.5.4 Commercial Standards**

- [5] ANSI/TIA/EIA-232-F, Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange, Revision F, 1997.
- [6] ANSI/TIA/EIA-422-B, Electrical Characteristics of Balanced Voltage Differential Interface Circuits, Revision B, 2005.
- [7] ANSI/TIA/EIA-423-B, Electrical Characteristics of Unbalanced Voltage Digital Interface Circuits, Revision B, 1996.

## A.2 SERVICES

### A.2.1 Provide Services

The *Serial Port Device API* Provide Service consists of the Table 1 service ports, interfaces, and primitives, which can be called by other client components. Detail definition of the interfaces and services shaded in gray is provided by separate JTRS API documentation referenced in the table. Operations detailed in gray are not applicable in the specified context.

Note: The *setRts()* operation is grayed out based on the assumption the Serial Port Device behaves as a DCE.

**Table 1 – Serial Port Device API Provide Service Interface**

Service Group (Port Name)	Service (Interface Provided)		Primitives (Provided)	Parameter Name or Return Value	Valid Range	
rx_data_provides_port	SerialPort::SerialPortPacketConsumer	Packet::FlowOctetStream [2]	pushPacket()	control	Packet::StreamControlType [2]	
				payload	JTRS::OctetSequence [1]	
				Return Value	TRUE or FALSE	
			Packet::PayloadStatus [2]	getMaxPayloadSize()	Return Value	See Appendix A.C.
				getMinPayloadSize()	Return Value	See Appendix A.C.
				getDesiredPayloadSize()	Return Value	See Appendix A.C.
		getMinOverrideTimeout()		Return Value	See Appendix A.C.	
		Packet::FlowControl [2]	enableFlowResumeSignals()	setEnabledState	TRUE or FALSE	
			spaceAvailable()	Return Value	TRUE or FALSE	
		Packet::EmptyControl [2]	enableEmptySignals()	setEnabledState	TRUE or FALSE	
		DeviceIo::DeviceIoControl [3]	enableRtsCts()	enable	TRUE or FALSE	
			setRts()	rtsToState	TRUE or FALSE	

Service Group (Port Name)	Service (Interface Provided)	Primitives (Provided)	Parameter Name or Return Value	Valid Range	
tx_flow_control_provides_port	SerialPort::SerialPortPacketProducer	Packet::PayloadControl [2]	setMaxPayloadSize()	maxPayloadSize	See Appendix A.C.
			setMinPayloadSize()	minPayloadSize	See Appendix A.C.
			setDesiredPayloadSize()	desiredPayloadSize	See Appendix A.C.
			setMinOverrideTimeout()	minOverrideTimeout	See Appendix A.C.
	Packet::FlowSignals [2]	signalResume()	<i>None</i>	<i>N/A</i>	
	Packet::EmptySignals [2]	signalEmpty()	streamId	See Appendix A.C.	
	DeviceIo::DeviceIoSignals [3]	setCts()	ctsToState	TRUE or FALSE	
config_provides_port	SerialPort::SerialPortConfiguration	setSynchronizationConfig()	See Section A.3, Service Primitives and Attributes	See Section A.3, Service Primitives and Attributes	
		getSynchronizationConfig()	See Section A.3, Service Primitives and Attributes	See Section A.3, Service Primitives and Attributes	
		getSynchronizationConfigsSupported()	See Section A.3, Service Primitives and Attributes	See Section A.3, Service Primitives and Attributes	
		setLoopback()	See Section A.3, Service Primitives and Attributes	See Section A.3, Service Primitives and Attributes	
		getLoopback()	See Section A.3, Service Primitives and Attributes	See Section A.3, Service Primitives and Attributes	
		setProtocolType()	See Section A.3, Service Primitives and Attributes	See Section A.3, Service Primitives and Attributes	

Service Group (Port Name)	Service (Interface Provided)		Primitives (Provided)	Parameter Name or Return Value	Valid Range
			getProtocolType()	See Section A.3, Service Primitives and Attributes	See Section A.3, Service Primitives and Attributes
msg_control_provides_port	SerialPort::SerialMessageControl	DevMsgCtl::DeviceMessageControl [4]	rxActive()	<i>Return Value</i>	TRUE or FALSE
			txActive()	<i>Return Value</i>	TRUE or FALSE
			abortTx()	<i>None</i>	N/A

## A.2.2 Provides Properties API

Table 2 provides the properties and their valid ranges that can be set and retrieved via the *CF::PropertySet::configure* and *CF::PropertySet::query* primitives.

**Table 2 – Serial Port Device Provide Properties**

<b>ID</b>	<b>Description</b>	<b>Type</b>	<b>Units</b>	<b>Valid Range</b>
SerialPortMaxPayloadSize	This property indicates the value of the Serial Port Consumer Maximum Payload Size	unsigned long	bytes	See Appendix A.C.
SerialPortMinPayloadSize	This property indicates the value of the Serial Port Consumer Minimum Payload Size	unsigned long	bytes	See Appendix A.C.
SerialPortDesiredPayloadSize	This property indicates the value of the Serial Port Consumer Desired Payload Size	unsigned long	bytes	See Appendix A.C.
SerialPortMinOverrideTimeOut	This property indicates the value of the Serial Port Consumer Minimum Override Timeout	unsigned long	ms	See Appendix A.C.
SerialPortEcho	This property provides the ability to enable/disable the Serial Port echo. If enabled, a linefeed is inserted after each carriage return for monitoring purposes only.	boolean	N/A	TRUE = enable; FALSE = disable
SerialASCII	This property provides the ability to enable/disable Serial Port Device ASCII. If enabled, for every carriage return detected, a linefeed character is inserted to the data stream pushed to the Device User. The linefeed is inserted for some terminals in order to display carriage returns correctly.	boolean	N/A	TRUE = enable; FALSE = disable

### A.2.3 Use Services

The *Serial Port Device API* Use Service set consists of the Table 3 service ports, interfaces, and primitives. Since a *Serial Port Device* component acts as a client with respect to these services from other components, it is required to connect these ports with corresponding service ports applied by the server component. Detail definition of the interfaces and services shaded in gray is provided by separate JTRS API documentation referenced in the table. Operations detailed in gray in Tabl3 3 are not applicable in the specified context.

Note: The setCts(), and enableRtsCts() operations are grayed out based on the assumption the SerialPort Device behaves as a DCE.

**Table 3 – Serial Port Device API Uses Service Interface**

Service Group (Port Name)	Service (Interface Provided)	Primitives (Provided)	Parameter Name or Return Value	Valid Range	
tx_data_uses_port	SerialPort::SerialPort Packet Consumer	Packet::FlowOctetStream [2]	pushPacket()	control	Packet::StreamControlType [2]
				payload	JTRS::OctetSequence [1]
				<i>Return Value</i>	TRUE or FALSE
		Packet::PayloadStatus [2]	getMaxPayloadSize()	<i>Return Value</i>	See Appendix A.C.
			getMinPayloadSize()	<i>Return Value</i>	See Appendix A.C.
			getDesiredPayloadSize()	<i>Return Value</i>	See Appendix A.C.
			getMinOverrideTimeout()	<i>Return Value</i>	See Appendix A.C.
		Packet::FlowControl [2]	enableFlowResumeSignals()	setEnabledState	TRUE or FALSE
			spaceAvailable()	<i>Return Value</i>	TRUE or FALSE
		Packet::EmptyControl [2]	enableEmptySignals()	setEnabledState	TRUE or FALSE
		DeviceIo::DeviceIoControl [3]	enableRtsCts()	enable	TRUE or FALSE
			setRts()	rtsToState	TRUE or FALSE
rx_flow_contr	SerialPort::	Packet::	setMaxPayloadSize()	maxPayloadSize	See Appendix A.C.

Service Group (Port Name)	Service (Interface Provided)		Primitives (Provided)	Parameter Name or Return Value	Valid Range
ol_uses_port	SerialPort Packet Producer	PayloadControl [2]	setMinPayloadSize()	minPayloadSize	See Appendix A.C.
			setDesiredPayloadSize()	desiredPayloadSize	See Appendix A.C.
			setMinOverrideTimeout()	minOverrideTimeout	See Appendix A.C.
		Packet:: FlowSignals [2]	signalResume()	<i>None</i>	<i>N/A</i>
		Packet:: EmptySignals [2]	signalEmpty()	streamId	See Appendix A.C.
		DeviceIo:: DeviceIoSignals [3]	setCts()	ctsToState	TRUE or FALSE

## A.2.4 Interface Modules

### A.2.4.1 SerialPort

The class diagram for the *Serial Port Device API* is shown in Figure 2. Interfaces shown in gray are defined in the *Packet API* [2], *DeviceIo API* [3], and *DeviceMessageControl API* [4].

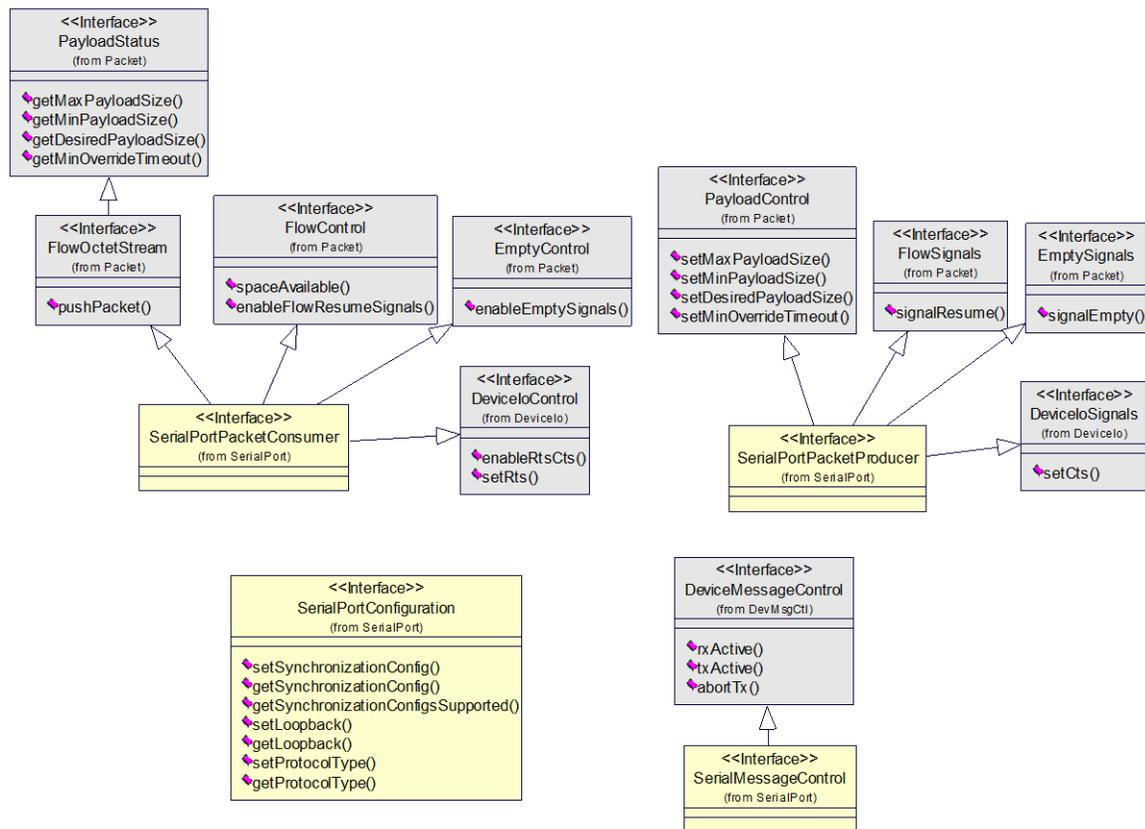


Figure 2 – Serial Port Device API Class Diagram

#### A.2.4.1.1 SerialPortConfiguration Interface Description

The interface design of the *SerialPortConfiguration* is shown in the Figure 3. It provides the capability to get and set the initial serial port configuration values.

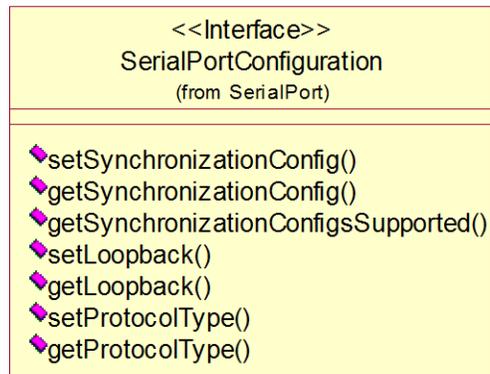


Figure 3 – SerialPortConfiguration Interface

### A.2.4.1.2 SerialPortPacketConsumer Interface Description

The interface design of the `SerialPortPacketConsumer` is shown in the interface class diagram. It extends the `FlowOctetStream`, `FlowControl`, `EmptyControl`, and `DeviceIoControl` interfaces to provide the capability to consume octet packets between the provider and user. It also supports RTS/CTS flow control. Interfaces shaded in gray are defined in *Packet API* [2] and *DeviceIo API* [3].

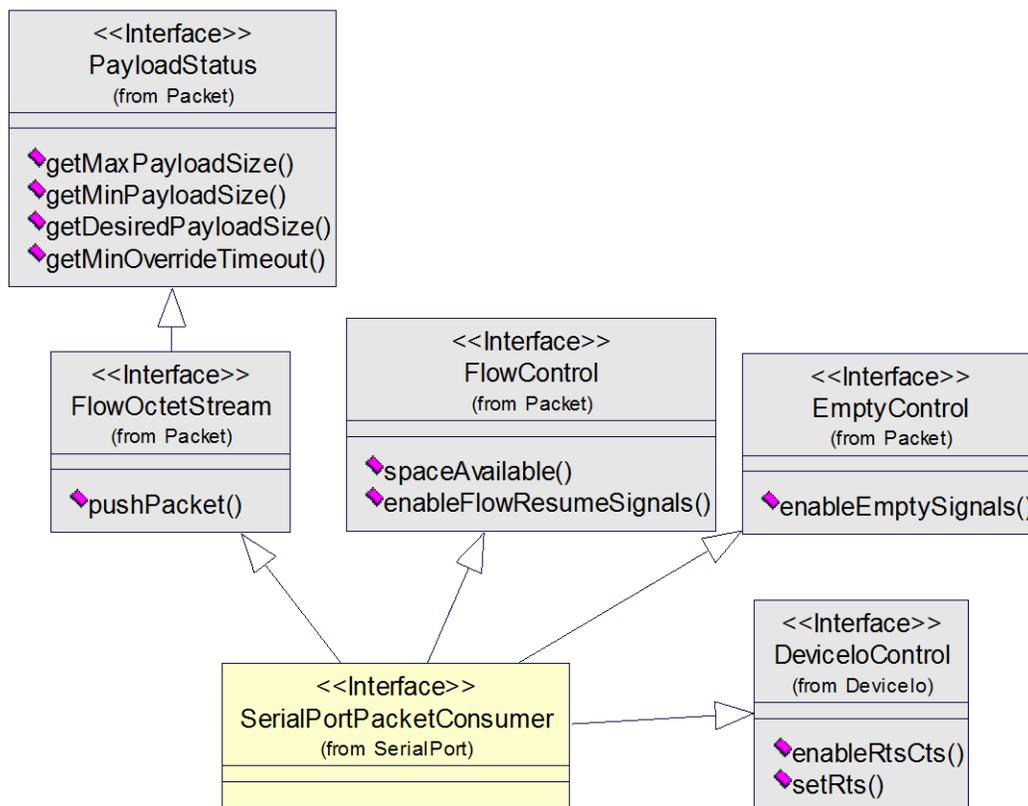


Figure 4 – SerialPortPacketConsumer Interface

### A.2.4.1.3 SerialPortPacketProducer Interface Description

The interface design of the *SerialPortPacketProducer* is shown in the interface class diagram. It extends the *PayloadControl*, *FlowSignals*, *EmptySignals* and *DeviceIoSignals* interfaces to provide the produce and control data from Serial Port HW. Interfaces shaded in gray are defined in *Packet API* [2] and *DeviceIo API* [3].

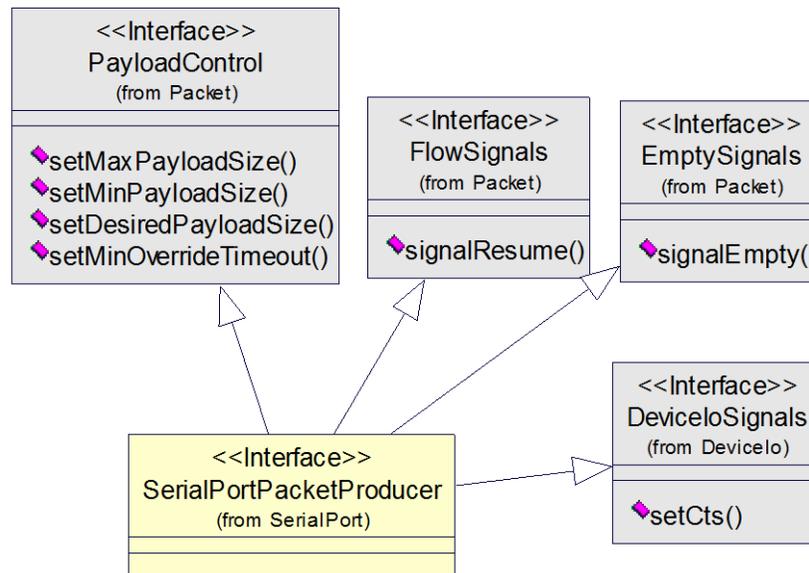


Figure 5 – SerialPortPacketProducer Interface

### A.2.4.1.4 SerialMessageControl Interface Description

The interface design of the *SerialMessageControl* is shown in Figure 6. It extends the *DeviceMessageControl* interface which provides the ability to control the device data flows. See *DeviceMessageControl API* [4].

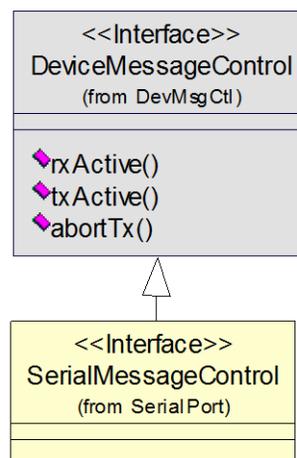


Figure 6 – SerialMessageControl Interface

## A.2.5 Sequence Diagrams

### A.2.5.1 Transfer Data

The Transfer Device Use Case covers scenarios in which data is transferred to or from a Serial Port Device. The scenarios are dependent on whether the Device needs to provide or consume data.

#### A.2.5.1.1 Device Provides Data Sequence Using Flow Resume and RTS/CTS Flow Control

##### Description

The *Device Provides Data* sequence diagram provides an example of how a *Device* provides data to the *Device User* using the flow resume and RTS/CTS flow controls. The *Device* in this sequence is the *Serial Port Device*. The *Device User* in the sequence is the *SerialPortPacketConsumer*. The *Device User* issues an *enableRtsCts()* and *enableFlowResumeSignals()* command to the *Device* to enable flow resume and RTS/CTS flow controls. When the *Device* receives incoming data from the *Device* HW, it sends a RTS<sup>1</sup> signal to the *Device User* indicating the start of a session. Once a CTS<sup>1,2</sup> signal is received, the *Device* pushes data received to the *Device User* using the *pushPacket* method. In this diagram, the flow control mechanism that can be used by the *Device User* to halt *pushPacket(s)* is demonstrated when the *pushPacket* returns FALSE. Once a *signalResume* is issued to the *Device*, the *Device* resumes pushing data to the *Device User* using the *pushPacket* method. See *Packet* [2] and *DeviceIo* [3] APIs for further details on the flow resume and RTS/CTS flow control operations.

Note: The following sequence diagram is based on the assumption the SerialPort Device behaves as a DCE.

##### Pre-conditions

None

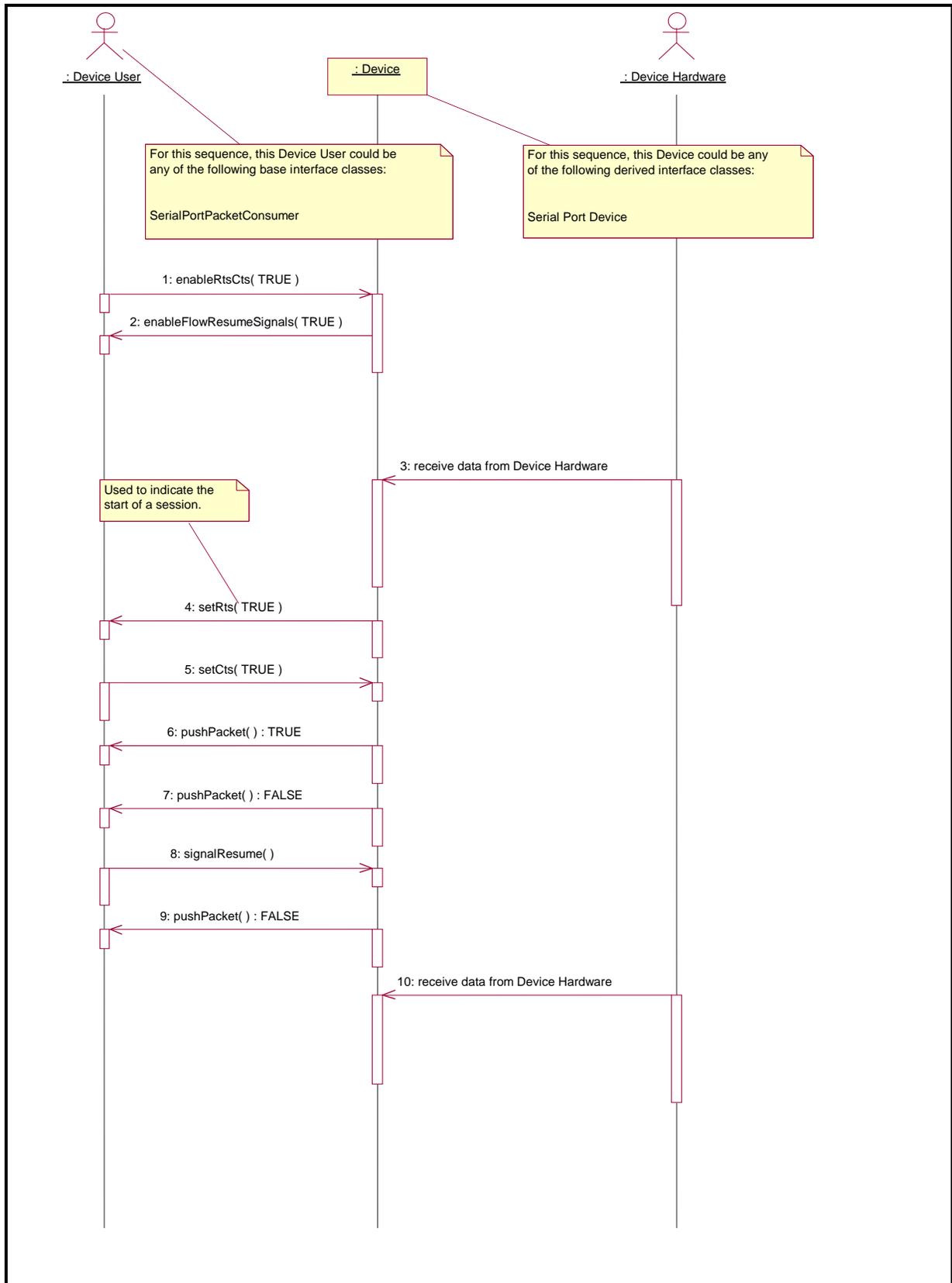
##### Post-conditions

The *Device* is pushing incoming data to the *Device User* and responding to flow control as needed.

---

<sup>1</sup> The RTS/CTS signals configured by the *setRts()* and *setCts()* operations may be tied directly to the RTS/CTS HW signals by configuring the *enableRtsCts* structure attribute defined within the associated synchronous configuration structure. See Sections C.5.4.1 and D.5.4.2.

<sup>2</sup> Alternate sequences of the RTS/CTS flow control protocol may be used. In certain cases where the available serial pins do not provide adequate information to control the state of the serial connection, and in-band signaling between the endpoints is used to supplement the normal RTS-CTS characteristics of the serial line, it is not necessary to wait for a CTS signal before sending control information.



**Figure 7 – Device Provides Data Sequence Diagram Using Flow Resume and RTS/CTS Flow Control**

### A.2.5.1.2 Device Consumes Data Sequence Using Flow Resume and RTS/CTS Flow Control

#### Description

The *Device Consumes Data* sequence diagram provides an example of how a Device consumes data provided by a DeviceUser (typical usage in a radio receiving data) using the flow resume. The *Device* in this sequence is *Serial Port Device*. The *Device User* in the sequence is the *SerialPortPacketProducer* interface (typically a waveform). The *Device User* issues an *enableFlowResumeSignals()* command to the *Device* to enable flow resume. The *Device User* pushes data to *Device* through the *pushPacket* method. In this diagram, the flow control mechanism that can be used by the *Device* to halt *pushPacket(s)* is demonstrated when the *pushPacket* returns FALSE. Once a *signalResume* signal is issued to the *Device User*, the *Device User* can continue pushing data to the *Device* using the *pushPacket* method. See *Packet* [2] and *DeviceIo* [3] APIs for further details on the flow resume and RTS/CTS flow controls.

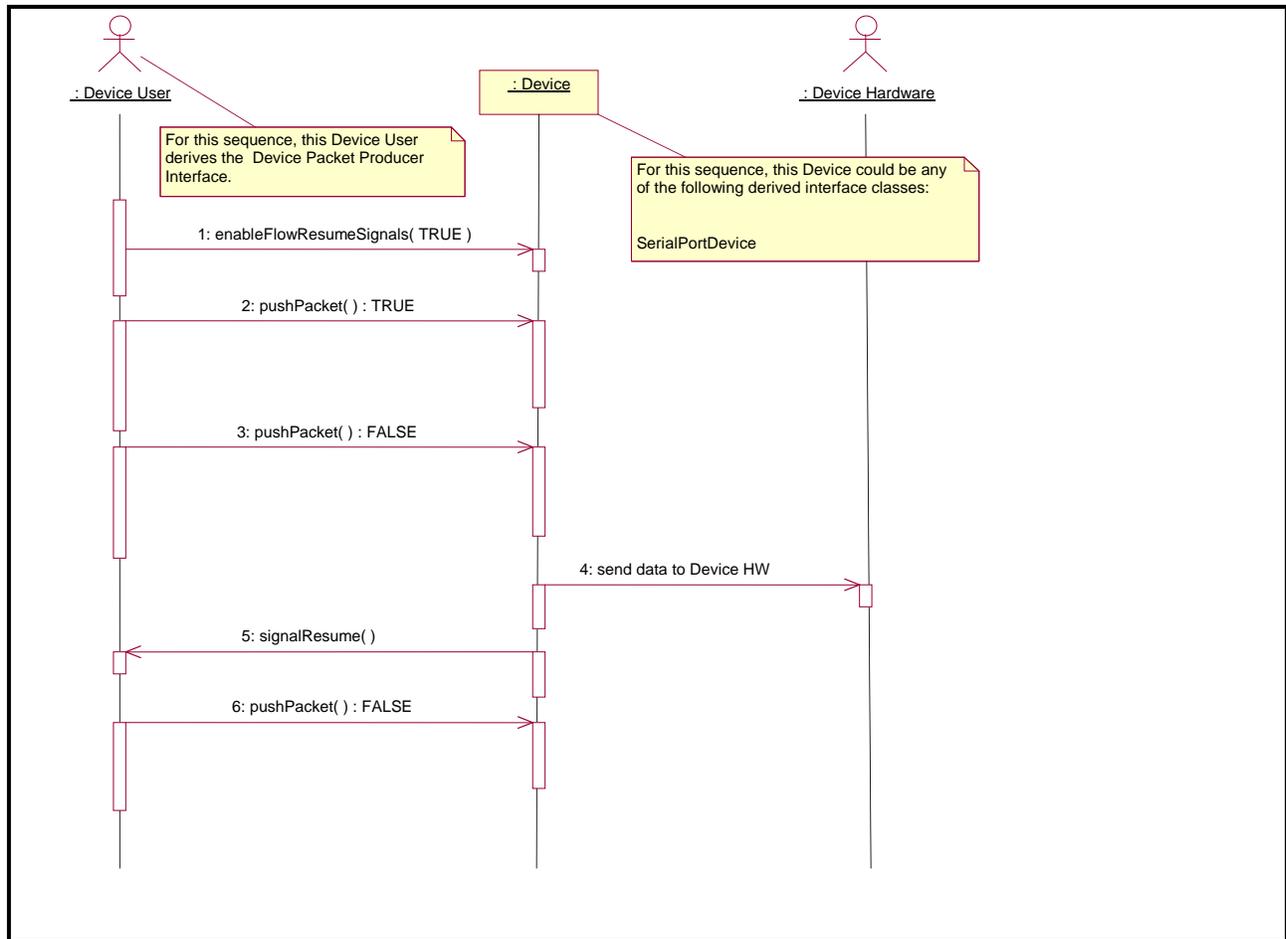
Note: The following sequence diagram is based on the assumption the SerialPort Device behaves as a DCE.

#### Pre-conditions

None

#### Post-conditions

The *Device User* is pushing outgoing data to the *Device* and responding to flow control signals as needed.



**Figure 8 – Device Consumes Data Sequence Diagram Using Flow Resume and RTS/CTS Flow Control**

## A.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the service primitives and attributes have been co-located in section A.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

### A.3.1 SerialPort::SerialPortPacketConsumer

The operations associated with the *SerialPortPacketConsumer* interface are documented in the *Packet* [2] and *DeviceIo* [3] APIs except specified as follows:

#### A.3.1.1 Packet::PayloadStatus::getDesiredPayloadSize Operation

The *getDesiredPayloadSize()* operation only applies to upstream data source from the Serial Port Device to the Device User, e.g. the SerialPortDevice in transmit. This parameter is unused in radio receive.

#### A.3.1.2 Packet::PayloadStatus::getMinOverrideTimeout Operation

The *getMinOverrideTimeout()* operation only applies to upstream data source from the Serial Port Device to the Device User, e.g. the SerialPortDevice in transmit. This parameter is unused in radio receive.

#### A.3.1.3 DeviceIo::DeviceIoControl::enableRtsCts Operation

The *enableRtsCts()* operation is used by the data consumer to specify if the data producer (e.g. Device in radio transmit) to indicate if CT is to be automatically returned by the data producer upon receipt of an RTS, or if the data source should only return CTS (and hold data packets) when the data consumer issues a *setCts()* operation.

#### A.3.1.4 DeviceIo::DeviceIoControl::setRts Operation

The RTS signal called by the *setRts()* operation may be tied directly to the RTS HW signal by configuring the *enableRtsCts* structure attribute defined within the associated synchronous configuration structure. See Sections C.5.4.1 and D.5.4.2.

### A.3.2 SerialPort::SerialPortPacketProducer

The operations associated with the *SerialPortPacketProducer* interface are documented in the *Packet* [2] and *DeviceIo* [3] APIs except as specified as follows.

#### A.3.2.1 Packet::PayloadStatus::setDesiredPayloadSize Operation

The *setDesiredPayloadSize()* operation only applies to downstream data source from the Device User to the Serial Port Device, e.g. the SerialPortDevice in receive. This parameter is unused in radio transmit.

#### A.3.2.2 Packet::PayloadStatus::setMinOverrideTimeout Operation

The *setDesiredPayloadSize()* operation only applies to downstream data source from the Device User to the Serial Port Device, e.g. the SerialPortDevice in receive. This parameter is unused in radio transmit.

### A.3.2.3 DeviceIo::DeviceIoSignals::setCts Operation

The CTS signal called by the *setCts()* operation may be tied directly to the CTS HW signal by configuring the *enableRtsCts* attribute defined within the associated synchronous configuration structure. See Sections C.5.4.1 and D.5.4.2.

## A.3.3 SerialPort::SerialMessageControl

The operations associated with the *SerialMessageControl* Interface are documented in the *Device Message Control [4] API*.

## A.3.4 SerialPort::SerialPortConfiguration

### A.3.4.1 setSynchronizationConfig Operation

The *setSynchronizationConfig* operation provides the ability to set *Serial Port Device* synchronization configuration type.

#### A.3.4.1.1 Synopsis

*void setSynchronizationConfig(in SynchronizationConfig synchronization) raises (JTRS::Unsupported);*

#### A.3.4.1.2 Parameters

Parameter Name	Type	Description
synchronization	SynchronizationConfig (See A.5.1.1)	The synchronization configuration of the <i>Serial Port Device</i> to be set.

#### A.3.4.1.3 State

Not applicable

#### A.3.4.1.4 New State

Not applicable

#### A.3.4.1.5 Return Value

None

#### A.3.4.1.6 Originator

Service User

#### A.3.4.1.7 Exceptions

Exception	Description
JTRS::Unsupported (See <i>JTRS CORBA Types [1]</i> )	The synchronization configuration selected is not supported by the <i>Serial Port Device</i> .

### **A.3.4.2 *getSynchronizationConfig* Operation**

The *getSynchronizationConfig* operation provides the ability to get *Serial Port Device* synchronization configuration type.

#### **A.3.4.2.1 Synopsis**

*SynchronizationConfig* *getSynchronizationConfig*();

#### **A.3.4.2.2 Parameters**

None

#### **A.3.4.2.3 State**

Not applicable

#### **A.3.4.2.4 New State**

Not applicable

#### **A.3.4.2.5 Return Value**

<b>Type</b>	<b>Description</b>
SynchronizationConfig (See A.5.1.1)	The current synchronization configuration of the <i>Serial Port Device</i> .

#### **A.3.4.2.6 Originator**

Service User

#### **A.3.4.2.7 Exceptions**

None

### **A.3.4.3 *getSynchronizationConfigsSupported* Operation**

The *getSynchronizationConfigsSupported* operation provides the ability to get all synchronization configuration types supported by the *Serial Port Device*.

#### **A.3.4.3.1 Synopsis**

*SynchronizationConfigSequence* *getSynchronizationConfigsSupported* ( );

#### **A.3.4.3.2 Parameters**

None

#### **A.3.4.3.3 State**

Not applicable

#### **A.3.4.3.4 New State**

Not applicable

#### **A.3.4.3.5 Return Value**

<b>Type</b>	<b>Description</b>
SynchronizationConfigSequence (See A.5.1.2)	A sequence containing all synchronization configurations supported by the <i>Serial Port Device</i> .

#### **A.3.4.3.6 Originator**

Service User

#### **A.3.4.3.7 Exceptions**

None

### A.3.4.4 *setLoopback* Operation

The *setLoopback* operation provides the ability to enable or disable the internal Serial Port HW loopback pin.

#### A.3.4.4.1 Synopsis

*void setLoopback(in boolean loopback);*

#### A.3.4.4.2 Parameters

Parameter Name	Description	Type	Units	Valid Range
loopback	The loopback value.	boolean	N/A	TRUE = enable FALSE = disable

#### A.3.4.4.3 State

Not applicable

#### A.3.4.4.4 New State

Not applicable

#### A.3.4.4.5 Return Value

None

#### A.3.4.4.6 Originator

Service User

#### A.3.4.4.7 Exceptions

None

### **A.3.4.5 *getLoopback* Operation**

The *getLoopback* operation provides the ability to get the status of loopback.

#### **A.3.4.5.1 Synopsis**

*boolean getLoopback();*

#### **A.3.4.5.2 Parameters**

None

#### **A.3.4.5.3 State**

Not applicable

#### **A.3.4.5.4 New State**

Not applicable

#### **A.3.4.5.5 Return Value**

<b>Type</b>	<b>Valid Range</b>	<b>Description</b>
boolean	TRUE	Loopback enabled.
	FALSE	Loopback disabled.

#### **A.3.4.5.6 Originator**

Service User

#### **A.3.4.5.7 Exceptions**

None

### A.3.4.6 *setProtocolType* Operation

The *setProtocolType* operation provides the ability to set the physical protocol type of the Serial Port HW as needed by a Device User.

#### A.3.4.6.1 Synopsis

*void setProtocolType(in ProtocolTypes protocol) raises (JTRS::InvalidParameter);*

#### A.3.4.6.2 Parameters

Parameter Name	Type	Description
protocol	ProtocolTypes (See A.5.2.1)	A protocol setting.

#### A.3.4.6.3 State

Not applicable

#### A.3.4.6.4 New State

Not applicable

#### A.3.4.6.5 Return Value

None

#### A.3.4.6.6 Originator

Service User

#### A.3.4.6.7 Exceptions

Type	Description
JTRS::InvalidParameter (See <i>JTRS CORBA Types</i> [1])	The protocol selected was invalid.

### **A.3.4.7 *getProtocolType* Operation**

The *getProtocolType* operation provides the ability to get current protocol type of the Serial Port HW.

#### **A.3.4.7.1 Synopsis**

*ProtocolTypes getProtocolType();*

#### **A.3.4.7.2 Parameters**

None

#### **A.3.4.7.3 State**

Not applicable

#### **A.3.4.7.4 New State**

Not applicable

#### **A.3.4.7.5 Return Value**

<b>Type</b>	<b>Description</b>
ProtocolTypes (See A.5.2.1)	The protocol settings.

#### **A.3.4.7.6 Originator**

Service User

#### **A.3.4.7.7 Exceptions**

None

## A.4 IDL

### A.4.1 SerialPort

```
/*
** SerialPort.idl
*/

#ifdef __SERIALPORT_DEFINED
#define __SERIALPORT_DEFINED

#ifdef __PACKET_DEFINED
#include "Packet.idl"
#endif
#ifdef __PACKETFLOWCONTROL_DEFINED
#include "PacketFlowControl.idl"
#endif
#ifdef __PACKETEMPTY SIGNALS_DEFINED
#include "PacketEmptySignals.idl"
#endif
#ifdef __DEVICEIO_DEFINED
#include "DeviceIo.idl"
#endif
#ifdef __JTRSCORBATYPES_DEFINED
#include "JtrsCorbaTypes.idl"
#endif
#ifdef __DEVICEMESSAGECONTROL_DEFINED
#include "DeviceMessageControl.idl"
#endif

module SerialPort {

    interface SerialPortPacketProducer : Packet::PayloadControl,
                                         Packet::FlowSignals,
                                         Packet::EmptySignals,
                                         DeviceIo::DeviceIoSignals
    {
    };

    interface SerialPortPacketConsumer : Packet::FlowControl,
                                         Packet::FlowOctetStream,
```

```
        Packet::EmptyControl,  
        DeviceIo::DeviceIoControl  
    {  
    };  
  
    interface SerialMessageControl : DevMsgCtl::DeviceMessageControl  
    {  
    };  
  
    typedef JTRS::ExtEnum          SynchronizationConfig;  
    typedef JTRS::ExtEnumSequence  SynchronizationConfigSequence;  
  
    // Base serial port constant  
    const SynchronizationConfig  CONFIG_NONE    = 0;  
  
    interface SerialPortConfiguration {  
  
        void setSynchronizationConfig (  
            in SynchronizationConfig synchronization  
        )  
            raises (JTRS::Unsupported);  
  
        SynchronizationConfig getSynchronizationConfig ();  
  
        SynchronizationConfigSequence getSynchronizationConfigsSupported ();  
  
        void setLoopback (  
            in boolean loopback  
        )  
            raises (JTRS::InvalidParameter);  
  
        boolean getLoopback ();  
  
        enum ProtocolTypes {  
            EIA_232,  
            EIA_422,  
            EIA_423  
        };  
  
        void setProtocolType (  

```

```
        in ProtocolTypes protocol
        )
        raises (JTRS::InvalidParameter);

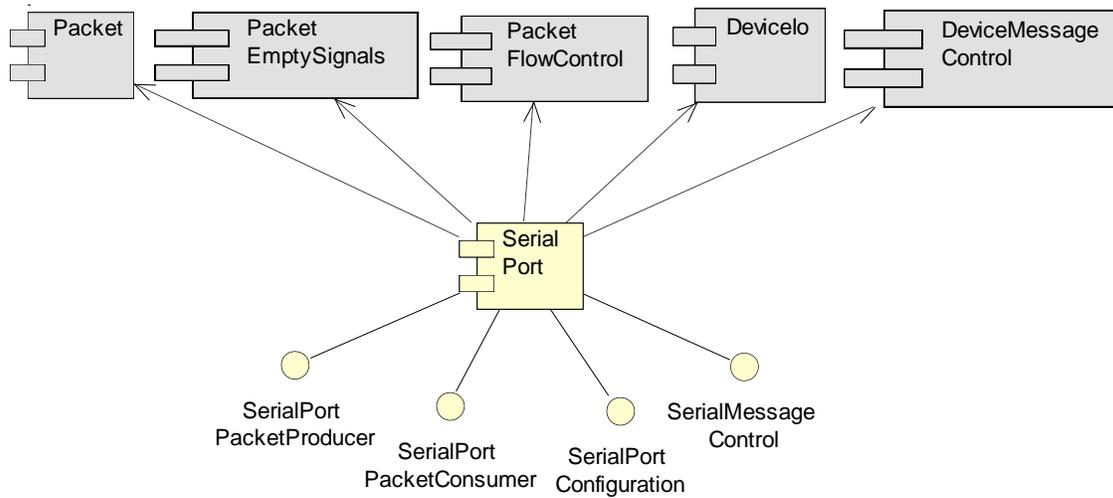
        ProtocolTypes getProtocolType ();

};

};
#endif // __SERIALPORT_DEFINED
```

## A.5 UML

This section contains the Device component UML diagram and the definitions of all data types referenced (directly or indirectly) by A.3 Service Primitives and Attributes.



**Figure 9 – Serial Port Device Component Diagram**

## A.5.1 Data Types

### A.5.1.1 SerialPort::SynchronizationConfig

The *SynchronizationConfig* type definition is a JTRS extension enumeration data type (see *JTRS CORBA Types* [1]). It enumerates the synchronization configurations supported by the *Serial Port Device*. Additional synchronization configurations supported by the *Serial Port Device* will be defined in their respective extensions.

```
typedef JTRS::ExtEnum          SynchronizationConfig;
const SynchronizationConfig    CONFIG_NONE = 0;
```

JTRS::Enum	Element	Value	Description
SynchronizationConfig	CONFIG_NONE	0	This is the default condition. The synchronization configuration has not been specified. No serial data should be transferred.

### A.5.1.2 SerialPort::SynchronizationConfigSequence

The *SynchronizationConfigSequence* is a JTRS extension enumeration sequence data type (see *JTRS CORBA Types* [1]). It is used to identify the set of synchronization configurations supported by the *Serial Port Device*.

```
typedef JTRS::ExtEnumSequence SynchronizationConfigSequence;
```

## A.5.2 Enumerations

### A.5.2.1 SerialPort::ProtocolTypes

The *ProtocolTypes* enumeration is used to set the protocol type to configure the voltage level on the Serial Port Device HW.

```
enum ProtocolTypes {
    EIA_232,
    EIA_422,
    EIA_423
};
```

Enum	Valid Range	Description
ProtocolTypes	EIA_232	EIA – 232 protocol. See <i>ANSI/TIA/EIA-232-F</i> [5].
	EIA_422	EIA – 422 protocol. See <i>ANSI/TIA/EIA-422-B</i> [6].
	EIA_423	EIA – 423 protocol. See <i>ANSI/TIA/EIA-423-B</i> [7].

## A.5.3 Exceptions

None

## **A.5.4 Structures**

None

## **APPENDIX A.A – ABBREVIATIONS AND ACRONYMS**

<b>API</b>	Application Program Interface
<b>CF</b>	Core Framework
<b>config</b>	configuration
<b>CORBA</b>	Common Object Request Broker Architecture
<b>CTS</b>	Clear To Send
<b>DCE</b>	Data Communications Equipment
<b>DTE</b>	Data Terminal Equipment
<b>EIA</b>	Electronic Industries Alliance
<b>HDLC</b>	High-Level Data Link Control
<b>HW</b>	Hardware
<b>ICWG</b>	Interface Control Working Group
<b>ID</b>	Identification
<b>IDL</b>	Interface Definition Language
<b>JPEO</b>	Joint Program Executive Office
<b>JTNC</b>	Joint Tactical Networking Center
<b>JTRS</b>	Joint Tactical Radio System
<b>ORB</b>	Object Request Broker
<b>RTS</b>	Request To Send
<b>SAR</b>	Software Anomaly Report
<b>UML</b>	Unified Modeling Language

## APPENDIX A.B – PERFORMANCE SPECIFICATION

Table 4 provides a template for the generic performance specification for the *Serial Port Device* which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 1.

**Table 4 – Serial Port Device Performance Specification**

Specification	Description	Units	Value
Worst Case Command Execution Time for pushPacket() on rx_data_provides_port	*	*	*
Worst Case Command Execution Time for rx_data_provides_port	*	*	*
Worst Case Command Execution Time for tx_flow_control_provides_port	*	*	*
Worst Case Command Execution Time for config_provides_port	*	*	*
Worst Case Command Execution Time for msg_control_provides_port	*	*	*
Worst Case Command Execution Time for tx_data_uses_port	*	*	*
Worst Case Command Execution Time for pushPacket() on tx_data_uses_port	*	*	*
Worst Case Command Execution Time for rx_flow_control_uses_port	*	*	*

Note: (\*) These values should be filled in by JTRS Product Line developers.

## APPENDIX A.C – PAYLOAD SIZE SPECIFICATION

Table 5 specifies a template for the valid ranges for the *Serial Port Device* payload size properties.

**Table 5 – Payload Size Properties Provide Service Interface**

ID	Description	Type	Units	Valid Range
SerialPortMaxPayloadSize	This property indicates the value of the SerialPort Consumer Maximum Payload Size	unsigned long	bytes	*
SerialPortMinPayloadSize	This property indicates the value of the SerialPort Consumer Minimum Payload Size	unsigned long	bytes	*
SerialPortDesiredPayloadSize	This property indicates the value of the SerialPort Consumer Desired Payload Size	unsigned long	bytes	*
SerialPortMinOverrideTimeOut	This property indicates the value of the Serial Port Consumer Minimum Override Timeout	unsigned long	ms	*

Note: (\*) These values should be filled in by JTRS Product Line developers.

Table 6 specifies a template for the valid ranges for the *Serial Port Device* payload size provide parameters or return values. Detail definition of the interfaces and services shaded in gray is provided by separate JTRS API documentation referenced in the table. Operations detailed in gray in Table 6 are not applicable in the specified context.

**Table 6 – Payload Size Provides Service Interface**

Service Group (Port Name)	Service (Interface Provided)		Primitives (Provided)	Parameter Name or Return Value	Valid Range	
rx_data_provides_port	SerialPort::SerialPortPacketConsumer	Packet::FlowOctetStream [2]	Packet::PayloadStatus [2]	getMaxPayloadSize()	<i>Return Value</i>	*
				getMinPayloadSize()	<i>Return Value</i>	*
				getDesiredPayloadSize()	<i>Return Value</i>	*
				getMinOverrideTimeout()	<i>Return Value</i>	*
tx_flow_contr	SerialPort::	Packet::	setMaxPayloadSize()	maxPayloadSize	*	

Service Group (Port Name)	Service (Interface Provided)		Primitives (Provided)	Parameter Name or Return Value	Valid Range
ol_provides_port	SerialPort Packet Producer	PayloadControl [2]	setMinPayloadSize()	minPayloadSize	*
			setDesiredPayloadSize()	desiredPayloadSize	*
			setMinOverrideTimeout()	minOverrideTimeout	*
	Packet::EmptySignals [2]	signalEmpty()	streamId	*	

Note: (\*) These values should be filled in by JTRS Product Line developers.

**Table 7 – Payload Size Uses Service Interface**

Service Group (Port Name)	Service (Interface Provided)			Primitives (Provided)	Parameter Name or Return Value	Valid Range
tx_data_uses_port	SerialPort::SerialPort Packet Consumer	Packet::FlowOctet Stream [2]	Packet::Payload Status [2]	getMaxPayloadSize()	<i>Return Value</i>	*
				getMinPayloadSize()	<i>Return Value</i>	*
				getDesiredPayloadSize()	<i>Return Value</i>	*
				getMinOverrideTimeout()	<i>Return Value</i>	*
rx_flow_control_uses_port	SerialPort::SerialPort Packet Producer	Packet::PayloadControl [2]	setMaxPayloadSize()	maxPayloadSize	*	
			setMinPayloadSize()	minPayloadSize	*	

Note: (\*) These values should be filled in by JTRS Product Line developers.

## B. ASYNCHRONOUS EXTENSION

### B.1 INTRODUCTION

The *Asynchronous (async) Extension* is based upon the *Serial Port Device API*. It extends the functionality of the Serial Port to include asynchronous serial capabilities.

#### B.1.1 Overview

This extension contains as follows:

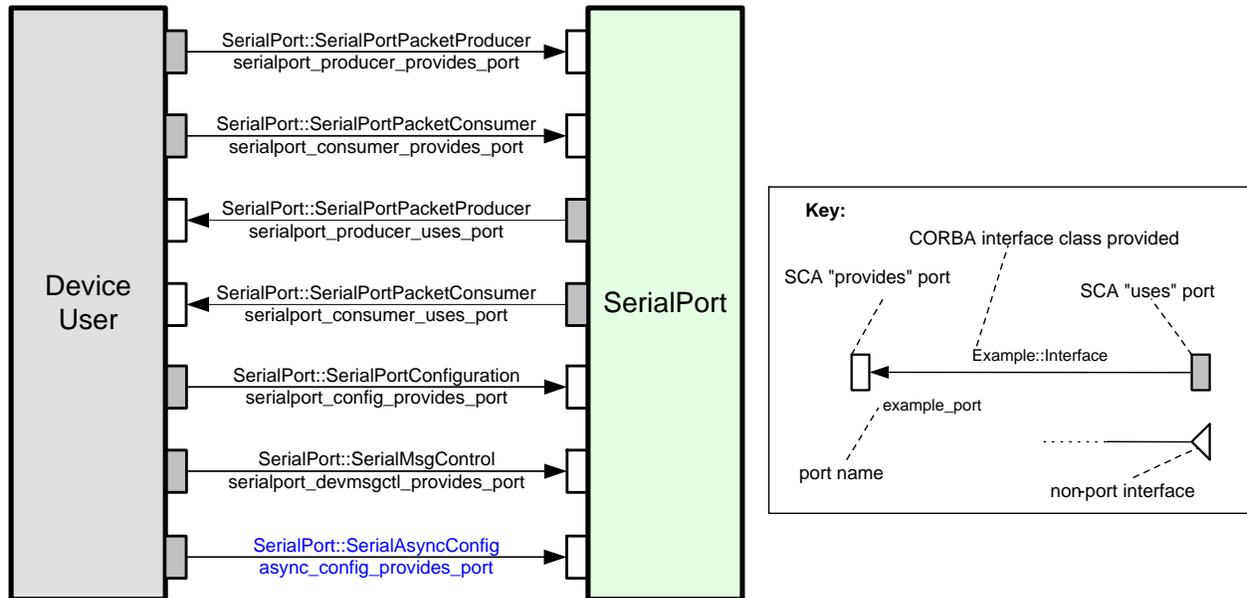
- a. Section B.1, *Introduction*, contains the introductory material regarding the overview, service layer description, modes, states and referenced documents of this document.
- b. Section B.2, *Services*, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.
- c. Section B.3, *Service Primitives and Attributes*, specifies the operations that are provided by the *Serial Port Device API Asynchronous Extension*.
- d. Section B.4, *IDL*.
- e. Section B.5, *UML*.
- f. Appendix B.A, – *Abbreviations and Acronyms*.
- i. Appendix B.B, – *Performance Specification*.
- j. Appendix B.C, – *Baud Rate Specification*

#### B.1.2 Service Layer Description

##### B.1.2.1 Asynchronous Serial Port Device API Extension Port Connections

Figure 10 shows the port connections for an asynchronous *Serial Port Device*.

Note: All port names are for reference only. Ports identified in black are provided in section A. Serial Port Device API.



**Figure 10 – Serial Port Device API Asynchronous Extension Port Diagram**

### ***Serial Port Device API Asynchronous Extension Provides Ports Definitions***

**async\_config\_provides\_port** is provided by the *Serial Port Device* to support the configuration of asynchronous serial.

### ***Serial Port Device API Asynchronous Extension Uses Ports Definitions***

None

## **B.1.3 Modes of Service**

Not applicable

## **B.1.4 Service States**

Not applicable

## **B.1.5 Referenced Documents**

There are no changes from section A. Serial Port Device API.

## B.2 SERVICES

### B.2.1 Provide Services

The *Asynchronous Extension* provides service consists of the Table 8 service ports, interfaces, and primitives, which can be called by other client components.

**Table 8 – Asynchronous Extension Provide Service Interface**

Service Group (Port Name)	Service (Interface Provided)	Primitives (Provided)
async_config_in_port	SerialPort::Async	setAsyncConfig()
		getAsyncConfig()

### B.2.2 Use Services

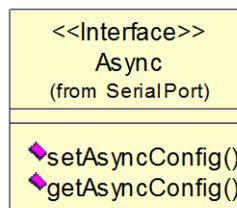
None

### B.2.3 Interface Modules

#### B.2.3.1 SerialPort

##### B.2.3.1.1 Asynchronous Extension Interface Description

The *Async* interface is shown in Figure 11. The *Async* interface provides the capability to set and get the asynchronous configuration.



**Figure 11 – Async Interface Class Diagram**

### B.2.4 Sequence Diagrams

None

## B.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the service primitives and attributes have been co-located in section B.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

### B.3.1 SerialPort::Async

#### B.3.1.1 *setAsyncConfig* Operation

The *setAsyncConfig* operation provides the ability to configure the asynchronous serial configuration.

##### B.3.1.1.1 Synopsis

*void setAsyncConfig (in AsyncConfigType asyncConfiguration) raises (JTRS::InvalidParameter);*

##### B.3.1.1.2 Parameters

Parameter Name	Type	Description
asyncConfiguration	AsyncConfigType (See B.5.4.1)	A structure containing the elements used for the initial configuration of asynchronous serial.

##### B.3.1.1.3 State

Not applicable

##### B.3.1.1.4 New State

Not applicable

##### B.3.1.1.5 Return Value

None

##### B.3.1.1.6 Originator

Service User

##### B.3.1.1.7 Exceptions

Type	Description
JTRS::InvalidParameter (See <i>JTRS CORBA Types</i> [1])	The configuration selected was invalid.

### **B.3.1.2 *getAsyncConfig* Operation**

#### **B.3.1.2.1 Synopsis**

*AsyncConfigType* *getAsyncConfig* ();

#### **B.3.1.2.2 Parameters**

None

#### **B.3.1.2.3 State**

Not applicable

#### **B.3.1.2.4 New State**

Not applicable

#### **B.3.1.2.5 Return Value**

<b>Type</b>	<b>Description</b>
AsyncConfigType (See B.5.4.1)	A structure containing the elements used for the initial configuration of asynchronous serial.

#### **B.3.1.2.6 Originator**

Service User

#### **B.3.1.2.7 Exceptions**

None

## B.4 IDL

### B.4.1 SerialAsyncExt

```
/*
** SerialAsyncExt.idl
*/

#ifndef __SERIALASYNCEXT_DEFINED
#define __SERIALASYNCEXT_DEFINED

#ifndef __SERIALPORT_DEFINED
#include "SerialPort.idl"
#endif

module SerialPort {

    const SynchronizationConfig    CONFIG_ASYNC    = CONFIG_NONE + 1;

    interface Async {

        /* The Parity enumerated type is used to establish the parity setting
        * in SerialPortDevice HW register. */

        enum ParitySet {

            /* Odd parity. */
            ODD_PARITY,
            /* Even parity. */
            EVEN_PARITY,
            /* No parity. */
            NO_PARITY
        };

        enum StopBitsType {

            /* Use to set one bit length. */
            ONE_BIT,
            /* Use to set one and a half bit length. */
            ONE_AND_HALF_BIT,
            /* Use to set two bit length. */
            TWO_BITS
        };

        enum NumberDataBitsType {

            /* Use to set the Five - bit data frame. */
            FIVE_BIT,
            /* Use to set the Six - bit data frame. */
            SIX_BIT,
            /* Use to set Seven - bit data frame. */
            SEVEN_BIT,
            /* Use to Eight - bit data frame. */
            EIGHT_BIT
        };

        struct AsyncConfigType {
            /* Set the number of stop bits. */

```

```
    StopBitsType numStopBits;
    /* Set the number data bit length. */
    NumberDataBitsType numDataBits;
    /* Set parity types (odd, even or no parity). */
    ParitySet parity;
    unsigned long baudRate;
    boolean hwFlowControl;
    boolean swFlowControl;
    boolean enableDma;
    /* This is a link status read only */
    boolean linkStatus;
};

void setAsyncConfig (
    in AsyncConfigType asyncConfiguration
) raises (JTRS::InvalidParameter);

AsyncConfigType getAsyncConfig ();

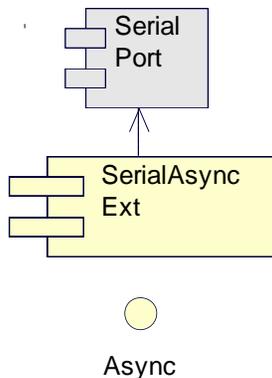
};

};

#endif // __SERIALASYNCEXT_DEFINED
```

## B.5 UML

This section contains the Device component UML diagram and the definitions of all data types referenced (directly or indirectly) by B.3 Service Primitives and Attributes.



**Figure 12 – Asynchronous Extension Component Diagram**

## B.5.1 Data Types

None

## B.5.2 Enumerations

### B.5.2.1 Async::NumberDataBitsType

The *NumberDataBitsType* enumeration is used to set the data word size in Serial Port HW control register.

```
enum NumberDataBitsType {
    FIVE_BIT,
    SIX_BIT,
    SEVEN_BIT,
    EIGHT_BIT
};
```

Enum	Attributes	Description
NumberDataBitsType	FIVE_BIT	Five bit data frame.
	SIX_BIT	Six bit data frame.
	SEVEN_BIT	Seven bit data frame.
	EIGHT_BIT	Eight bit data frame.

### B.5.2.2 Async::ParitySet

The *ParitySet* enumeration is used to establish the parity setting in the Serial Port HW register.

```
enum ParitySet {
    ODD_PARITY,
    EVEN_PARITY,
    NO_PARITY
};
```

Enum	Attributes	Description
ParitySet	ODD_PARITY	odd parity
	EVEN_PARITY	even parity
	NO_PARITY	no parity

### B.5.2.3 Async::StopBitsType

The *StopBitsType* enumeration is used to set the stop bit for a transmitted data.

```
enum StopBitsType {
    ONE_BIT,
    ONE_AND_HALF_BIT,
    TWO_BITS
};
```

Enum	Attributes	Description
StopBitsType	ONE_BIT	one bit length
	ONE_AND_HALF_BIT	one and a half bit length
	TWO_BITS	two bit lengths

### B.5.2.4 SerialPort::SynchronizationConfig

The following specifies an additional synchronization configuration supported by the *Serial Port Device API* that is defined in this extension.

```
const SynchronizationConfig CONFIG_ASYNC = CONFIG_NONE + 1;
```

JTRS::Enum	Element	Value	Description
SynchronizationConfig (See A.5.1.1)	CONFIG_ASYNC	CONFIG_NONE+1	Identifies the asynchronous synchronization configuration

## B.5.3 Exceptions

None

## B.5.4 Structures

### B.5.4.1 Async::AsyncConfigType

The *AsyncConfigType* structure defines the configurable properties of asynchronous synchronization.

```
struct AsyncConfigType {
    StopBitsType numStopBits;
    NumberDataBitsType numDataBits;
    ParitySet parity;
    unsigned long baudRate;
    boolean hwFlowControl;
    boolean swFlowControl;
    boolean enableDma;
    boolean linkStatus;
};
```

<b>Struct</b>	<b>Attributes</b>	<b>Type</b>	<b>Valid Range</b>	<b>Description</b>
AsyncConfigType	numStopBits	StopBitsType	See B.5.2.3	This attribute indicates the number of stop bits.
	numDataBits	NumberDataBitsType	See B.5.2.1	This attribute indicates the data bit length.
	parity	ParitySet	See B.5.2.2	This attribute indicates the parity type.
	baudRate	unsigned long	See Appendix B.C	This attribute indicates the baud rate (in bps) used for Serial Port HW.
	hwFlowControl*	boolean	TRUE=enable; FALSE=disable	This attribute is used to indicate HW flow control.
	swFlowControl*	boolean	TRUE=enable (Xon – Resume data transmission/Xoff – Pause data transmission); FALSE=disable	This attribute is used as a software flow control mechanism to indicate whether or not data is ready to be received.
	enableDma*	boolean	TRUE=enable; FALSE=disable	This attribute indicates whether Direct Memory Access (DMA) is enabled.
	linkStatus*	boolean	TRUE=enable; FALSE=disable	This is a read only link status attribute.

Note(\*): Optional configuration features may not be supported by all *Serial Port Devices*. These configurations may be disabled if not needed.

## APPENDIX B.A – ABBREVIATIONS AND ACRONYMS

This following lists additional abbreviations and acronyms not specified in section A. Serial Port Device API.

<b>async</b>	asynchronous
<b>bps</b>	bits per second
<b>DMA</b>	Direct Memory Access
<b>SW</b>	software
<b>X</b>	transmitter

## APPENDIX B.B – PERFORMANCE SPECIFICATION

Table 9 provides a template for the generic performance specification for the *Asynchronous (async) Extension Serial Port Device* API which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 10.

**Table 9 – Serial Port Device Performance Specification**

Specification	Description	Units	Value
Worst Case Command Execution Time for <code>async_config_provides_port</code>	*	*	*

Note: (\*) These values should be filled in by JTRS Product Line developers.

## APPENDIX B.C – BAUD RATE SPECIFICATION

Table 10 specifies a template to document the valid ranges for the *Asynchronous (async) Extension* baud rate attribute.

**Table 10 – Baud Rate Specification**

Struct	Attributes	Type	Valid Range	Description
AsyncConfigType (See Section B.5.4.1)	baudRate	unsigned long	*	This attribute indicates the baud rate (in bps) used for Serial Port HW.

Note: (\*) These values should be filled in by JTRS Product Line developers.

## C. SYNCHRONOUS RAW EXTENSION

### C.1 INTRODUCTION

The *Synchronous (sync) Raw Extension* is based upon the *Serial Port Device API*. It extends the functionality of the Serial Port to include synchronous raw serial capabilities.

#### C.1.1 Overview

This extension contains as follows:

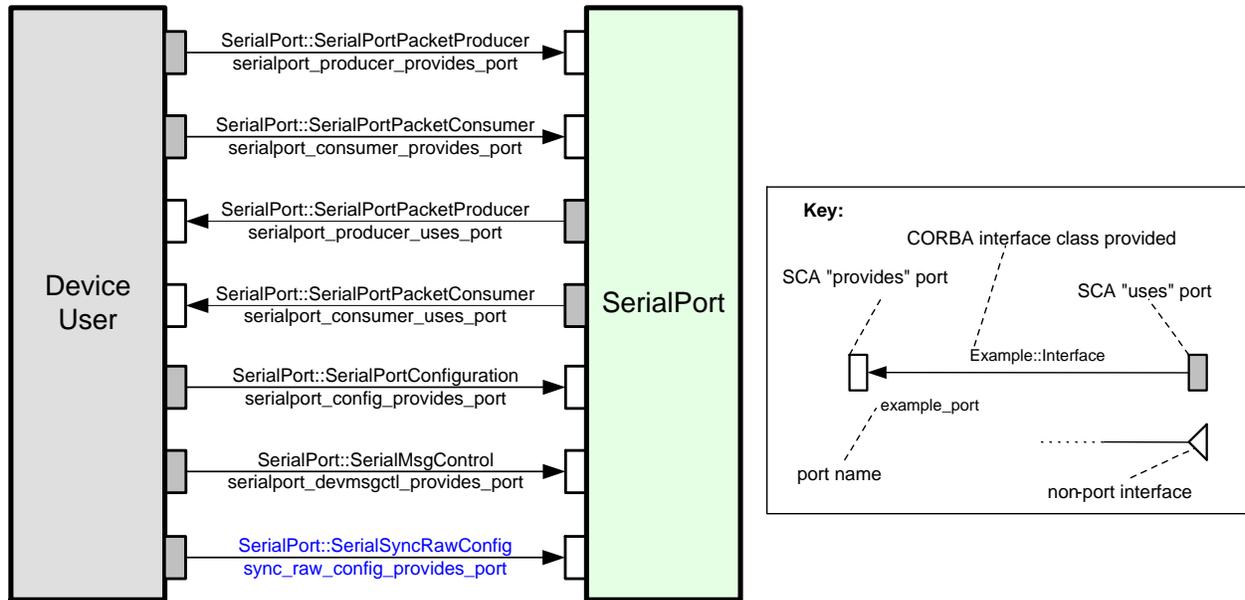
- a. Section C.1, *Introduction*, , contains the introductory material regarding the overview, service layer description, modes, states and referenced documents of this document.
- b. Section C.2, *Services*, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.
- c. Section C.3, *Service Primitives and Attributes*, specifies the operations that are provided by the *Serial Port Device API Synchronous Raw Extension*.
- d. Section C.4, *IDL*.
- e. Section C.5, *UML*.
- f. Appendix C.A, – Abbreviations and Acronyms.
- g. Appendix C.B, – *Performance Specification*.
- h. Appendix C.C, – *Baud Rate Specification*.
- i. Appendix C.D, – *Automatic Baud Rate Limit Specification*.

#### C.1.2 Service Layer Description

##### C.1.2.1 Synchronous Raw Serial Port Device API Extension Port Connections

Figure 13 shows the port connections for a synchronous raw *Serial Port Device*.

Note: All port names are for reference only. Ports identified in black are provided in section A. Serial Port Device API.



**Figure 13 – Serial Port Device API Synchronous Raw Extension Port Diagram**

**Serial Port Device API Synchronous Raw Extension Provides Ports Definitions**

`sync_raw_config_provides_port` is provided by the *Serial Port Device* to support the configuration of synchronous raw serial.

**Serial Port Device API Synchronous Raw Extension Uses Ports Definitions**

None

**C.1.3 Modes of Service**

Not applicable

**C.1.4 Service States**

Not applicable

**C.1.5 Referenced Documents**

There are no changes from section A. Serial Port Device API.

## C.2 SERVICES

### C.2.1 Provide Services

The *SynchronousRaw Extension* provides service consists of the Table 11 service ports, interfaces, and primitives, which can be called by other client components.

**Table 11 – Synchronous Raw Extension Provide Service Interface**

Service Group (Port Name)	Service (Interface Provided)	Primitives (Provided)
sync_raw_config_provides_port	SerialPort::SyncRaw	setSyncRawConfig()
		getSyncRawConfig()

### C.2.2 Use Services

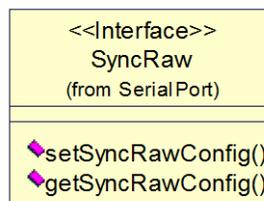
None

### C.2.3 Interface Modules

#### C.2.3.1 SerialPort

##### C.2.3.1.1 Synchronous Raw Extension Interface Description

The *SyncRaw* interface is shown in Figure 14. The *SyncRaw* interface provides the capability to set and get the synchronous raw configuration.



**Figure 14 – SyncRaw Interface Class Diagram**

### C.2.4 Sequence Diagrams

None

## C.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the service primitives and attributes have been co-located in section C.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

### C.3.1 SerialPort::SyncRaw

#### C.3.1.1 *setSyncRawConfig* Operation

The *setSyncRawConfig* operation provides the ability to configure the synchronous raw serial configuration.

##### C.3.1.1.1 Synopsis

*void setSyncRawConfig (in SyncRawConfigType syncRawConfiguration)  
raises (JTRS::InvalidParameter);*

##### C.3.1.1.2 Parameters

Parameter Name	Type	Description
syncRawConfiguration	SyncRawConfigType (See C.5.4.1)	A structure containing the elements used for the initial configuration of synchronous raw serial.

##### C.3.1.1.3 State

Not applicable

##### C.3.1.1.4 New State

Not applicable

##### C.3.1.1.5 Return Value

None

##### C.3.1.1.6 Originator

Service User

##### C.3.1.1.7 Exceptions

Type	Description
JTRS::InvalidParameter (See <i>JTRS CORBA Types</i> [1])	The configuration selected was invalid.

### C.3.1.2 *getSyncRawConfig* Operation

#### C.3.1.2.1 Synopsis

*SyncRawConfigType* *getSyncRawConfig*();

#### C.3.1.2.2 Parameters

None

#### C.3.1.2.3 State

Not applicable

#### C.3.1.2.4 New State

Not applicable

#### C.3.1.2.5 Return Value

Type	Description
SyncRawConfigType (See C.5.4.1)	A structure containing the elements used for the initial configuration of synchronous raw serial.

#### C.3.1.2.6 Originator

Service User

#### C.3.1.2.7 Exceptions

None

## C.4 IDL

### C.4.1 SerialSyncRawExt

```
/*
** SerialSyncRawExt.idl
*/

#ifndef __SERIALSYNCRRAWEXT_DEFINED
#define __SERIALSYNCRRAWEXT_DEFINED

#ifndef __SERIALPORT_DEFINED
#include "SerialPort.idl"
#endif

module SerialPort {

    const SynchronizationConfig CONFIG_SYNC_RAW = CONFIG_NONE + 2;

    interface SyncRaw{

        enum ClockSources {

            /* Use to set the clock source which supports by the SerialPort
            * HW. */
            DCE,
            /* Use to set the clock source which supports by the SerialPort
            * HW. */
            DTE
        };

        enum SyncClockControlType {
            /* Transmit clock is on when the RTS and CTS HW signals are both high */
            RTS_CTS,
            /* Transmit clock is free running */
            FREE_RUNNING
        };

        struct SyncRawConfigType {
            octet autoBaudLimit;
            /* Indicates type of transmitted Clock Source is set for the
```

```
        * synchronization Configuration. */
        ClockSources txClockSource;
        SyncClockControlType clockCtrl;
        unsigned long baudRate;
        boolean enableRtsCts;
        boolean enableCdpMode;
    };

    void setSyncRawConfig (
        in SyncRawConfigType syncRawConfiguration
    )
        raises (JTRS::InvalidParameter);

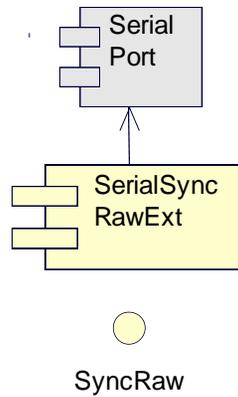
    SyncRawConfigType getSyncRawConfig ();
};

};

#endif // __SERIALSYNCRRAWEXT_DEFINED
```

## C.5 UML

This section contains the Device component UML diagram and the definitions of all data types referenced (directly or indirectly) by C.3 Service Primitives and Attributes.



**Figure 15 – Synchronous Raw Extension Component Diagram**

## C.5.1 Data Types

None

## C.5.2 Enumerations

### C.5.2.1 SerialPort::SynchronizationConfig

The following specifies an additional synchronization configuration supported by the *Serial Port Device API* that is defined in this extension.

```
const SynchronizationConfig CONFIG_SYNC_RAW = CONFIG_NONE + 2;
```

JTRS::Enum	Element	Value	Description
SynchronizationConfig (See A.5.1.1)	CONFIG_SYNC_RAW	CONFIG_NONE+2	Identifies the synchronous raw synchronization configuration

### C.5.2.2 SyncRaw::ClockSources

The *ClockSources* enumeration is used to set the transmit clock sources supported by the Serial Port HW.

```
enum ClockSources {
    DCE,
    DTE
};
```

Enum	Attributes	Description
ClockSources	DCE	Data Communications Equipment (DCE) Clock, use to set the internal transmit clock source for the SerialPort HW.
	DTE	Data Terminal Equipment (DTE) Clock, use to set the external transmit clock source for the SerialPort HW.

### C.5.2.3 SyncRaw::SyncClockControlType

The *SyncClockControlType* enumeration is used to configure the clock control type for the Serial Port.

```
enum SyncClockControlType {
    RTS_CTS,
    FREE_RUNNING
};
```

Enum	Valid Range	Description
SyncClockControlType	RTS_CTS	Transmit clock is on when the RTS and CTS HW signals are both high
	FREE_RUNNING	Transmit clock is free running

## C.5.3 Exceptions

None

## C.5.4 Structures

### C.5.4.1 SyncRaw::SyncRawConfigType

The *SyncRawConfigType* structure defines the configurable properties of synchronous raw synchronization.

```
struct SyncRawConfigType {
    octet autoBaudLimit;
    ClockSources txClockSource;
    SyncClockControlType clockCtrl;
    unsigned long baudRate;
    boolean enableRtsCts;
    boolean enableCdpMode;
};
```

Struct	Attributes	Type	Valid Range	Description
SyncRawConfigType	autoBaudLimit*	octet	zero value = disable automatic baud rate adjustment;  non-zero value = enable automatic baud rate adjustment. This value will be the maximum allowable adjustment of the configured baud rate (see Appendix C.D).	This attribute enables/disables automatic baud rate adjustment. If enabled, the value specifies the automatic baud rate adjustment limit based on the rate at which the <i>Device User</i> pushes data (+/- n% of nominal baud rate).
	txClockSource	ClockSources	See C.5.2.2	This attribute indicates the type of clock.
	clockCtrl	SyncClock ControlType	See C.5.2.3	This attribute sets the HW clock control type.
	baudRate	unsigned long	See Appendix C.C	This attribute indicates the configured baud rate used for Serial Port HW.
	enableRtsCts	boolean	TRUE= enable; FALSE=disable	This attribute determines whether the RTS/CTS control signals configured by <i>setRts()</i> and <i>setCts()</i> are tied directly to the RTS/CTS HW signals. See A.3.1.4 and A.3.2.3.
enableCdpMode*	boolean	TRUE = enable; FALSE = disable	This attribute enables Conditional Di-Phase (CDP) encoding.	

Note(\*): Optional configuration features may not be supported by all *Serial Port Devices*. These configurations may be disabled if not needed.

## APPENDIX C.A – ABBREVIATIONS AND ACRONYMS

This following lists additional abbreviations and acronyms not specified in section A. Serial Port Device API.

<b>CDP</b>	Continuous Di-Phase
<b>sync</b>	synchronous
<b>tx</b>	transmit

## APPENDIX C.B – PERFORMANCE SPECIFICATION

Table 12 provides a template for the generic performance specification for the *Synchronous (sync) Raw Extension Serial Port Device* which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 13.

**Table 12 – Serial Port Device Performance Specification**

Specification	Description	Units	Value
Worst Case Command Execution Time for sync_raw_config_provides_port	*	*	*

Note: (\*) These values should be filled in by JTRS Product Line developers.

## APPENDIX C.C – BAUD RATE SPECIFICATION

Table 13 specifies a template to document the valid ranges for *Synchronous (sync) Raw Extension* the baud rate attribute.

**Table 13 – Baud Rate Specification**

Struct	Attributes	Type	Valid Range	Description
SyncRawConfigType (See Section C.5.4.1)	baudRate	unsigned long	*	This attribute indicates the baud rate (in bps) used for Serial Port HW.

Note: (\*) These values should be filled in by JTRS Product Line developers.

## APPENDIX C.D – AUTOMATIC BAUD RATE LIMIT SPECIFICATION

Table 14 specifies a template to document the valid range for the automatic baud rate limit associated with the *Synchronous (sync) Raw Extension autoBaudLimit* attribute.

**Table 14 – Automatic Baud Rate Limit Specification**

Struct	Attributes	Type	Valid Range	Description
SyncRawConfigType (See Section C.5.4.1)	autoBaudLimit	octet	*	This attribute sets the automatic baud rate adjustment limit based on the rate at which the <i>Device User</i> pushes data (+/- percentage of nominal baud rate).

Note: (\*) These values should be filled in by JTRS Product Line developers.

## D. SYNCHRONOUS HDLC EXTENSION

### D.1 INTRODUCTION

The *Synchronous (sync) High-Level Data Link Control (HDLC) Extension* is based upon the *Serial Port Device API*. It extends the functionality of the Serial Port to include synchronous serial capabilities using HDLC.

#### D.1.1 Overview

This extension contains as follows:

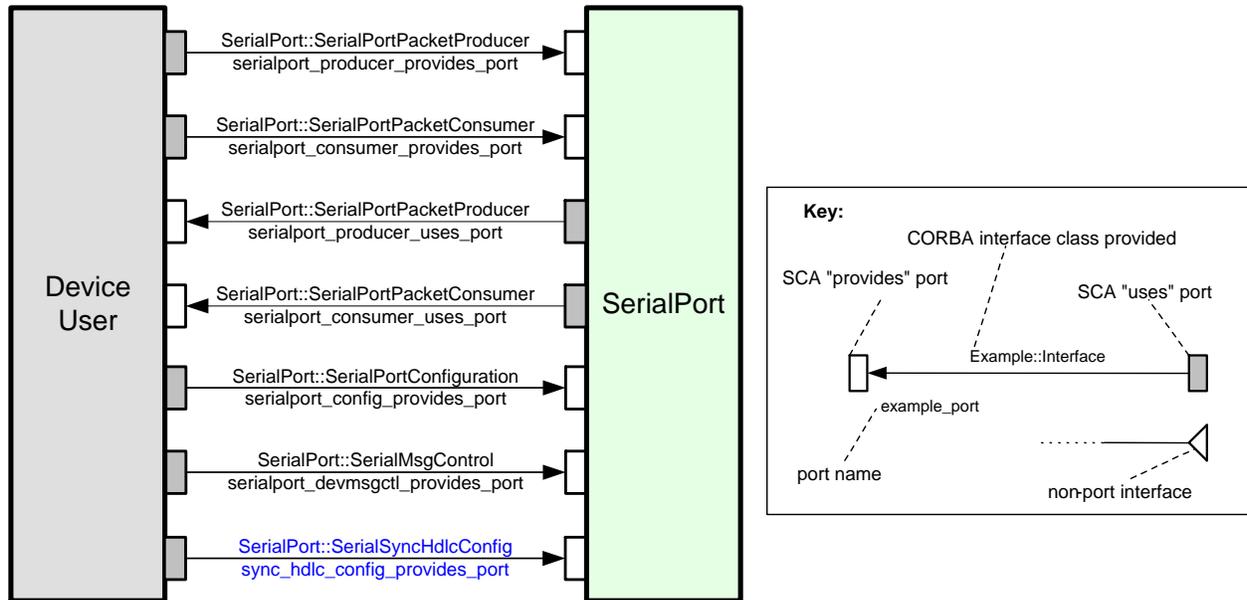
- a. Section D.1, *Introduction*, contains the introductory material regarding the overview, service layer description, modes, states and referenced documents of this document.
- b. Section D.2, *Services*, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.
- c. Section D.3, *Service Primitives* and Attributes, specifies the operations that are provided by the *Serial Port Device API Synchronous HDLC Extension*.
- d. Section D.4, *IDL*.
- e. Section D.5, *UML*.
- f. Appendix D.A, – *Abbreviations and Acronyms*.
- g. Appendix D.B, – *Performance Specification*.
- h. Appendix D.C, – *Baud Rate Specification*.
- i. Appendix D.D, – *Automatic Baud Rate Limit Specification*.

#### D.1.2 Service Layer Description

##### D.1.2.1 Synchronous HDLC Serial Port Device API Extension Port Connections

Figure 16 shows the port connections for a synchronous *Serial Port Device* using HDLC.

Note: All port names are for reference only. Ports identified in black are provided in section A. Serial Port Device API.



**Figure 16 – Serial Port Device API Synchronous HDLC Extension Port Diagram**

**Serial Port Device API Synchronous HDLC Extension Provides Ports Definitions**

`serialport_sync_hdlc_config_provides_port` is provided by the *Serial Port Device* to support the configuration of synchronous serial using HDLC.

**Serial Port Device API Synchronous HDLC Extension Uses Ports Definitions**

None

**D.1.3 Modes of Service**

Not applicable

**D.1.4 Service States**

Not applicable

**D.1.5 Referenced Documents**

There are no changes from section A. Serial Port Device API.

## D.2 SERVICES

### D.2.1 Provide Services

The *Synchronous HDLC Extension* provides service consists of the Table 15 service ports, interfaces, and primitives, which can be called by other client components.

**Table 15 – Synchronous HDLC Extension Provide Service Interface**

Service Group (Port Name)	Service (Interface Provided)	Primitives (Provided)
sync_hdlc_config_provides_port	SerialPort::SyncHdlc	setSyncHdlcConfig()
		getSyncHdlcConfig()

### D.2.2 Use Services

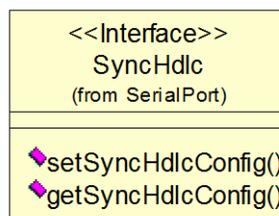
None

### D.2.3 Interface Modules

#### D.2.3.1 SerialPort

##### D.2.3.1.1 SyncHdlc Extension Interface Description

The *SyncHdlc* interface is shown in Figure 17. The *SyncHdlc* interface provides the capability to set and get the synchronous serial port using HDLC configuration.



**Figure 17 – SyncHdlc Interface Class Diagram**

### D.2.4 Sequence Diagrams

None

## D.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the service primitives and attributes have been co-located in section D.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

### D.3.1 SerialPort::SyncHdlc

#### D.3.1.1 *setSyncHdlcConfig* Operation

The *setSyncHdlcConfig* operation provides the ability to configure the synchronous serial port using HDLC.

##### D.3.1.1.1 Synopsis

```
void setSyncHdlcConfig ( in SyncHdlcConfigType syncHdlcConfiguration)
    raises (JTRS::InvalidParameter);
```

##### D.3.1.1.2 Parameters

Parameter Name	Type	Description
syncHdlcConfiguration	SyncHdlcConfigType (See D.5.4.2)	A structure containing the elements used for the initial configuration of the synchronous serial port using HDLC.

##### D.3.1.1.3 State

Not applicable

##### D.3.1.1.4 New State

Not applicable

##### D.3.1.1.5 Return Value

None

##### D.3.1.1.6 Originator

Service User

##### D.3.1.1.7 Exceptions

Type	Description
JTRS::InvalidParameter (See <i>JTRS CORBA Types</i> [1])	The configuration selected was invalid.

### **D.3.1.2 *getSyncHdlcConfig* Operation**

#### **D.3.1.2.1 Synopsis**

*SyncHdlcConfigType* *getSyncHdlcConfig* ();

#### **D.3.1.2.2 Parameters**

None

#### **D.3.1.2.3 State**

Not applicable

#### **D.3.1.2.4 New State**

Not applicable

#### **D.3.1.2.5 Return Value**

<b>Type</b>	<b>Description</b>
SyncHdlcConfigType (See D.5.4.2)	A structure containing the elements used for the initial configuration of synchronous serial using HDLC.

#### **D.3.1.2.6 Originator**

Service User

#### **D.3.1.2.7 Exceptions**

None

## D.4 IDL

### D.4.1 SerialSyncHdlcExt

```
/*
** SerialSyncHdlcExt.idl
*/

#ifndef __SERIALSYNCHDLCEXT_DEFINED
#define __SERIALSYNCHDLCEXT_DEFINED

#ifndef __SERIALPORT_DEFINED
#include "SerialPort.idl"
#endif

module SerialPort {

    const SynchronizationConfig CONFIG_SYNC_HDLC = CONFIG_NONE + 3;

    interface SyncHdlc{

        enum ClockSources {
            /* Use to set the clock source which supports by the SerialPort
            * HW. */
            DCE,
            /* Use to set the clock source which supports by the SerialPort
            * HW. */
            DTE
        };

        enum SyncClockControlType {
            /* Transmit clock is on when the RTS and CTS HW signals are both high */
            RTS_CTS,
            /* Transmit clock is free running */
            FREE_RUNNING
        };

        struct HdlcHeaderType {
            /* Set the number of headers. */
            octet numOfHeaders;
            /* Set header 0. */
        };
    };
};
```

```
    octet header0;
    /* Set header 1. */
    octet header1;
    /* Set header 2. */
    octet header2;
    /* Set header 3. */
    octet header3;
    /* Set header 4. */
    octet header4;
    /* Set header 5. */
    octet header5;
};

struct SyncHdlcConfigType {
    unsigned long baudRate;

    /* Indicates type of transmitted Clock Source is set for the
    * synchronization Configuration. */

    ClockSources txClockSource;

    HdlcHeaderType txHdlcHeader;
    HdlcHeaderType rxHdlcHeader;

    SyncClockControlType clockCtrl;

    boolean txCrcEnable;
    boolean rxCrcEnable;

    boolean txCrcLength;
    boolean rxCrcLength;

    boolean txAltFrameFill;
    boolean rxAltFrameFill;

    octet autoBaudLimit;
    boolean enableRtsCts;
    boolean enableCdpMode;

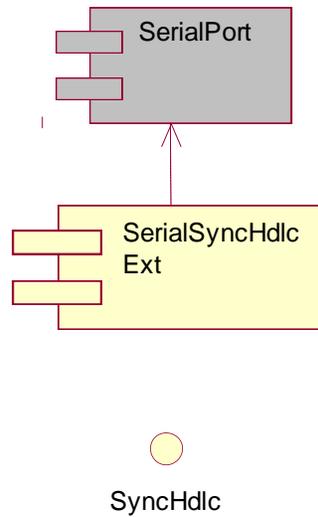
};
```

```
        void setSyncHdlcConfig (
            in SyncHdlcConfigType syncHdlcConfiguration
        )
            raises (JTRS::InvalidParameter);

        SyncHdlcConfigType getSyncHdlcConfig ();
    };
};
#endif // __SERIALSYNCHDLCEXT_DEFINED
```

## D.5 UML

This section contains the Device component UML diagram and the definitions of all data types referenced (directly or indirectly) by D.3 Service Primitives and Attributes.



**Figure 18 – Synchronous HDLC Extension Component Diagram**

## D.5.1 Data Types

None

## D.5.2 Enumerations

### D.5.2.1 SerialPort::SynchronizationConfig

The following specifies an additional synchronization configuration supported by the *Serial Port Device API* that is defined in this extension.

```
const SynchronizationConfig CONFIG_SYNC_HDLIC = CONFIG_NONE + 3;
```

JTRS::Enum	Element	Value	Description
SynchronizationConfig (See A.5.1.1)	CONFIG_SYNC_HDLIC	CONFIG_NONE+3	Identifies the synchronous synchronization configuration using HDLC

### D.5.2.2 SyncHdlc::ClockSources

The *ClockSources* enumeration is used to set the transmit clock sources supported by the Serial Port HW.

```
enum ClockSources {
    DCE,
    DTE
};
```

Enum	Attributes	Description
ClockSources	DCE	DCE Clock, use to set the internal transmit clock source for the Serial Port HW.
	DTE	DTE Clock, use to set the external transmit clock source for the Serial Port HW.

### D.5.2.3 SyncHdlc::SyncClockControlType Enumeration

The *SyncClockControlType* enumeration is used to configure the clock control type for the Serial Port.

```
enum SyncClockControlType {
    RTS_CTS,
    FREE_RUNNING
};
```

Enum	Valid Range	Description
SyncClockControlType	RTS_CTS	Transmit clock is on when the RTS and CTS HW signals are both high
	FREE_RUNNING	Transmit clock is free running

## D.5.3 Exceptions

None

## D.5.4 Structures

### D.5.4.1 SyncHdlc::HdlcHeaderType Structure

The *HdlcHeaderType* structure is used to indicate the HDLC header for Serial Port HW.

```
struct HdlcHeaderType {
    octet numOfHeaders;
    octet header0;
    octet header1;
    octet header2;
    octet header3;
    octet header4;
    octet header5;
};
```

Enum	Attributes	Type	Valid Range	Description
HdlcHeaderType	numOfHeaders	octet	0 - 6	Indicates the number of HDLC header that is used.
	header0	octet	0 - 255	
	header1	octet	0 - 255	
	header2	octet	0 - 255	
	header3	octet	0 - 255	
	header4	octet	0 - 255	
	header5	octet	0 - 255	

### D.5.4.2 SyncHdlc::SyncHdlcConfigType Structure

The *SyncHdlcConfigType* structure defines the configurable properties of synchronous serial using HDLC synchronization.

```
struct SyncHdlcConfigType {
    unsigned long baudRate;
    ClockSources txClockSource;
    HdlcHeaderType txHdlcHeader;
    HdlcHeaderType rxHdlcHeader;
    SyncClockControlType clockCtrl;
    boolean txCrcEnable;
    boolean rxCrcEnable;
    boolean txCrcLength;
    boolean rxCrcLength;
    boolean txAltFrameFill;
    boolean rxAltFrameFill;
    octet autoBaudLimit;
    boolean enableRtsCts;
    boolean enableCdpMode;
};
```

Struct	Attributes	Type	Valid Range	Description
SyncHdlc ConfigType	baudRate	unsigned long	See Appendix D.C	This attribute indicates the baud rate is used for Serial Port HW.

Struct	Attributes	Type	Valid Range	Description
	txClockSource	ClockSources	See D.5.2.2	This attribute indicates the type of clock.
	txHdlcHeader	HdlcHeaderType	See D.5.4.1	This attribute indicates the HDLC header
	rxHdlcHeader	HdlcHeaderType	See D.5.4.1	This attribute indicates the HDLC header.
	clockCtrl	SyncClockControlType	See D.5.2.3	This attribute sets the HW clock control type.
	txCrcEnable	boolean	TRUE= enable; FALSE= disable	This attribute enables tx Cyclic Redundancy Checking (CRC) generation.
	rxCrcEnable	boolean	TRUE= enable; FALSE= disable	This attribute enables rx CRC generation.
	txCrcLength	boolean	TRUE= 32 bits; FALSE=16 bits	This attribute sets the tx CRC length.
	rxCrcLength	boolean	TRUE= 32 bits; FALSE=16 bits	This attribute sets the rx CRC length.
	txAltFrameFill	boolean	TRUE=0xFF; FALSE=0x7E	This attribute sets the tx interframe flag fill.
	rxAltFrameFill	boolean	TRUE=0xFF; FALSE=0x7E	This attribute sets the rx interframe flag fill.
	autoBaudLimit*	octet	zero value = disable automatic baud rate adjustment;  non-zero value = enable automatic baud rate adjustment. This value will be the maximum allowable adjustment of the configured baud rate (see Appendix D.D).	This attribute enables/disables automatic baud rate adjustment. If enabled, the value specifies the automatic baud rate adjustment limit based on the rate at which the <i>Device User</i> pushes data (+/- n% of nominal baud rate).
	enableRtsCts	boolean	TRUE= enable; FALSE=disable	This attribute determines whether the RTS/CTS control signals configured by <i>setRts()</i> and <i>setCts()</i> are tied directly to the RTS/CTS HW signals. See A.3.1.4 and A.3.2.3.
	enableCdpMode*	boolean	TRUE = enable; FALSE = disable	This attribute enables Conditional Di-Phase (CDP) encoding.

Note(\*): Optional configuration features may not be supported by all *Serial Port Devices*. These configurations may be disabled if not needed.

## APPENDIX D.A – ABBREVIATIONS AND ACRONYMS

This following lists additional abbreviations and acronyms not specified in A. Serial Port Device API.

<b>CRC</b>	Cyclic Redundancy Checking
<b>CTS</b>	Clear To Send
<b>DCE</b>	Data Communications Equipment
<b>DTC</b>	Data Terminal Equipment
<b>HDLC</b>	High-level Data Link Control
<b>RTS</b>	Request To Send
<b>Rx</b>	Receive
<b>Sync</b>	synchronous
<b>Tx</b>	Transmit

## APPENDIX D.B – PERFORMANCE SPECIFICATION

Table 16 provides a template for the generic performance specification for the *Synchronous (sync) High-Level Data Link Control (HDLC) Extension Serial Port Device API* which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 16.

**Table 16 – Serial Port Device Performance Specification**

Specification	Description	Units	Value
Worst Case Command Execution Time for <code>sync_hdlc_config_provides_port</code>	*	*	*

Note: (\*) These values should be filled in by JTRS Product Line developers.

## APPENDIX D.C – BAUD RATE SPECIFICATION

Table 17 specifies a template to document the valid ranges for the *Synchronous (sync) High-Level Data Link Control (HDLC) Extension* baud rate attribute.

**Table 17 – Baud Rate Specification**

Struct	Attributes	Type	Valid Range	Description
SyncHdlcConfigType (See Section D.5.4.2)	baudRate	unsigned long	*	This attribute indicates the baud rate (in bps) used for Serial Port HW.

Note: (\*) These values should be filled in by JTRS Product Line developers.

## APPENDIX D.D – AUTOMATIC BAUD RATE LIMIT SPECIFICATION

Table 18 specifies a template to document the valid range for the automatic baud rate limit associated with the *Synchronous (sync) High-Level Data Link Control (HDLC) Extension autoBaudLimit* attribute.

**Table 18 – Automatic Baud Rate Limit Specification**

<b>Struct</b>	<b>Attributes</b>	<b>Type</b>	<b>Valid Range</b>	<b>Description</b>
SyncHdlcConfigType (See Section D.5.4.2)	autoBaudLimit	octet	*	This attribute sets the automatic baud rate adjustment limit based on the rate at which the <i>Device User</i> pushes data.

Note: (\*) These values should be filled in by JTRS Product Line developers.

## E. DYNAMIC BAUD EXTENSION

### E.1 INTRODUCTION

The *Dynamic Baud Extension* is based upon the *Serial Port Device API*. It extends the functionality of the base *Serial Port Device* to support the dynamic configuration of the baud rate.

#### E.1.1 Overview

This extension contains as follows:

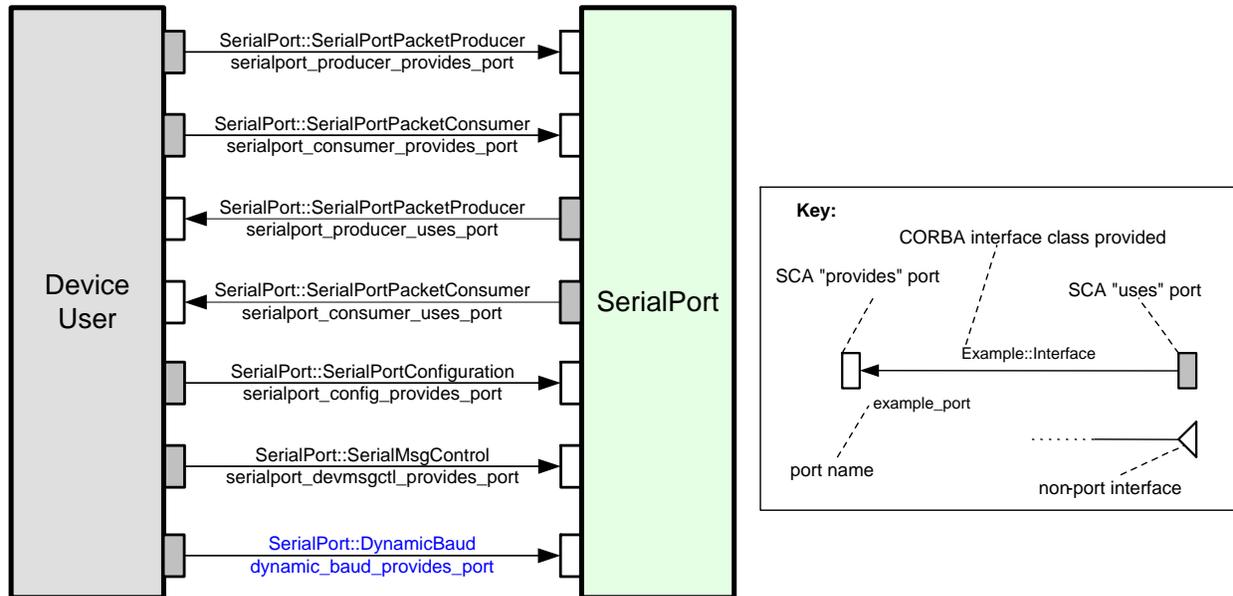
- a. Section E.1, *Introduction*, of this document contains the introductory material regarding the overview, Service Layer Description, Modes, States and Referenced Documents of this document.
- b. Section E.2, *Services*, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.
- c. Section E.3, *Service Primitives and Attributes*, specifies the operations that are provided by the *Serial Port Device API Dynamic Baud Extension*.
- d. Section E.4, *IDL*.
- e. Section E.5, *UML*.
- f. Appendix E.A, – *Abbreviations and Acronyms*.
- g. Appendix E.B, – *Performance Specification*.

#### E.1.2 Service Layer Description

##### E.1.2.1 Dynamic Baud Serial Port Device API Extension Port Connections

The following figure shows the port connections for the *Serial Port Device API Dynamic Baud Extension*. Note: All port names are for reference only.

Note: All port names are for reference only. Ports identified in black are provided in section A. Serial Port Device API.



**Figure 19 – Serial Port Device API Dynamic Baud Extension Port Diagram**

**Serial Port Device API Dynamic Baud Extension Provides Ports Definitions**

**dynamic\_baud\_provides\_port** is provided by the *Serial Port Device* to support the dynamic configuration of the baud rate.

**Serial Port Device API Dynamic Baud Extension Uses Ports Definitions**

None

**E.1.3 Modes of Service**

Not applicable

**E.1.4 Service States**

Not applicable

**E.1.5 Referenced Documents**

There are no changes from the base API.

## E.2 SERVICES

### E.2.1 Provide Services

The *Dynamic Baud Extension* provides service consists of the following service ports, interfaces, and primitives, which can be called by other client components.

**Table 19 – Dynamic Baud Extension Provide Service Interface**

Service Group (Port Name)	Service (Interface Provided)	Primitives (Provided)
dynamic_baud_provides_port	SerialPort::DynamicBaud	setBaudRate()
		getEffectiveBaudRate()

### E.2.2 Use Services

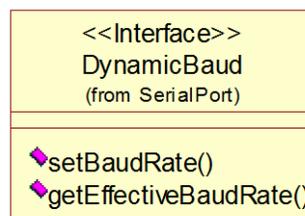
None

### E.2.3 Interface Modules

#### E.2.3.1 SerialPort

##### E.2.3.1.1 Dynamic Baud Extension Interface Description

The *DynamicBaud* interface is shown in the interface class diagram below. The *DynamicBaud* interface provides the capability to set the baud rate and get the effective baud rate.



**Figure 20 – Dynamic Baud Interface Class Diagram**

### E.2.4 Sequence Diagrams

None

## E.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., data types, enumerations, exceptions, and structures) used by the Service Primitives and Attributes have been co-located in section E.5, UML. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

### E.3.1 SerialPort::DynamicBaud

#### E.3.1.1 *setBaudRate* Operation

The *setBaudRate* operation allows the user to set the baud rate without the *Device* changing its state. The *Device* will make a “best effort” attempt at the rate adjustment and adjust to the HW supported rate which is nearest to the request rate.

##### E.3.1.1.1 Synopsis

*unsigned long setBaudRate (in DataFlow direction, in unsigned long baudRate )*  
*raises (JTRS::InvalidParameter);*

##### E.3.1.1.2 Parameters

Parameter Name	Type	Units	Description
direction	DataFlow (See Section E.5.2.1)	N/A	The direction in which to adjust the rate.
baudRate	unsigned long	bps	The new baud rate in units of 0.01 bits per second (bps).

##### E.3.1.1.3 State

Not applicable

##### E.3.1.1.4 New State

Not applicable

##### E.3.1.1.5 Return Value

Type	Units	Description
unsigned long	bps	The effective baud rate in units of 0.01 bps.

##### E.3.1.1.6 Originator

Device User

##### E.3.1.1.7 Exceptions

Type	Description
JTRS::InvalidParameter (see JTRS CORBA Types [1])	The <i>InvalidParameter</i> exception indicates that the requested baud rate is invalid.

### **E.3.1.2 *getEffectiveBaudRate* Operation**

The *getEffectiveBaudRate* operation allows the user to get the current effective baud rate.

#### **E.3.1.2.1 Synopsis**

*void getEffectiveBaudRate (out unsigned long txBaudRate, out unsigned long rxBaudRate);*

#### **E.3.1.2.2 Parameters**

<b>Parameter Name</b>	<b>Type</b>	<b>Units</b>	<b>Description</b>
txBaudRate	unsigned long	bps	The effective baud rate in units of 0.01 bps in the waveform OTA transmit direction.
rxBaudRate	unsigned long	bps	The effective baud rate in units of 0.01 bps in the waveform OTA receive direction.

#### **E.3.1.2.3 State**

Not applicable

#### **E.3.1.2.4 New State**

Not applicable

#### **E.3.1.2.5 Return Value**

None

#### **E.3.1.2.6 Originator**

Device User

#### **E.3.1.2.7 Exceptions**

None

## E.4 IDL

### E.4.1 SerialDynBaudExt

```
/*
** SerialDynBaudExt.idl
*/

#ifndef __SERIALDYNBAUDEXT_DEFINED
#define __SERIALDYNBAUDEXT_DEFINED

#ifndef __SERIALPORT_DEFINED
#include "SerialPort.idl"
#endif

/* SerialPortDevice */

module SerialPort {

    interface DynamicBaud {

        enum DataFlow {
            TX,      // Waveform OTA TX
            RX,      // Waveform OTA RX
            BOTH
        };

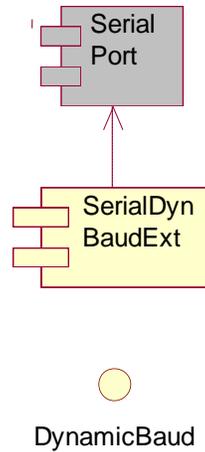
        unsigned long setBaudRate (
            in DataFlow direction,
            in unsigned long baudRate // in 0.01bps
        ) raises( JTRS::InvalidParameter );

        void getEffectiveBaudRate (
            out unsigned long txBaudRate, // Waveform OTA TX in 0.01bps
            out unsigned long rxBaudRate // Waveform OTA RX in 0.01bps
        );
    };
};

#endif // __SERIALDYNBAUDEXT_DEFINED
```

## E.5 UML

This section contains the Device component UML diagram and the definitions of all data types referenced (directly or indirectly) by the Service Primitives and Attributes in E.3.



**Figure 21 – Dynamic Baud Extension Component Diagram**

## E.5.1 Data Types

None

## E.5.2 Enumerations

### E.5.2.1 DynamicBaud::DataFlow

The *DataFlow* enumeration is used to give the direction in which to adjust the rate.

```
enum DataFlow {
    TX,
    RX,
    BOTH    };
```

Enum	Attributes	Description
DataFlow	TX	Waveform OTA transmit data flow direction
	RX	Waveform OTA receive data flow direction
	Both	Both transmit and receive data flow directions

## E.5.3 Exceptions

None

## E.5.4 Structures

None

## APPENDIX E.A – ABBREVIATIONS AND ACRONYMS

This following lists additional abbreviations and acronyms not specified in section A. Serial Port Device API.

<b>bps</b>	bits per second
<b>OTA</b>	Over The Air
<b>Rx</b>	Receive
<b>Tx</b>	Transmit

## APPENDIX E.B – PERFORMANCE SPECIFICATION

Table 20 provides a template for the generic performance specification for the *Dynamic Baud Extension Serial Port Device* which will be documented in the service or device using the interface. This performance specification corresponds to the port diagram in Figure 19

**Table 20 – Serial Port Device Performance Specification**

Specification	Description	Units	Value
Worst Case Command Execution Time for <code>dynamic_baud_provides_port</code>	*	*	*

Note: (\*) These values should be filled in by JTRS Product Line developers.